NO. 33 $2.00 FEBRUARY 1981

MICRO™

THE 6502 JOURNAL

In the Heart of Applesoft

PET String Flip

Increase KIM-1 Versatility at Low Cost

An Atari Assembler

A C1P Sound Idea

A Simple Securities Manager for the Apple

NOW THE SOFTCARD™ CAN TAKE YOU BEYOND THE BASICS.

You probably know about the SoftCard — our ingenious circuit card that converts an Apple II® into a Z-80® machine running CP/M®.

You may even know that with the SoftCard, you get Microsoft's powerful BASIC — extended to support Apple graphics and many other features.

Now, whenever you're ready to get beyond the BASICs, the SoftCard can take you into whole new realms. Starting with two advanced language packages from Microsoft.

FORTRAN AND COBOL TO GO.

Now you can run the world's most popular engineering/scientific language and the most popular business language on your Apple. Think what that means: you can choose from literally thousands of "off-the-shelf" applications programs, and have them working with little conversion. Or design your own programs, taking advantage of all the problem-solving power these specialized languages give you.

FORTRAN-80

A complete ANSI-standard FORTRAN (except COMPLEX type), with important enhancements. The extremely fast compiler performs extensive code optimization, and, since it doesn't require a "P-code" interpreter at run time, your programs will typically execute 2-3 times faster than with Apple FORTRAN.

FORTRAN is easy to learn if you know BASIC, and the package includes a huge library of floating point, math, and I/O routines you can use in all your programs.

COBOL-80

Virtually the only choice for serious business data processing. It's ANSI 1974 standard COBOL, with many user-oriented features added: formatted screen support for CRT terminals, simple segmenting of very large programs, powerful file handling capability, trace debugging, and much more. A separate Sort package is coming soon.

FORTRAN-80 and COBOL-80 are just two more reasons why the Apple with SoftCard is the world's most versatile personal computer. Get all the exciting details from your Microsoft dealer today. And start getting beyond the BASICs.

MICROSOFT Consumer Products, 400 108th Ave. N.E., Suite 200, Bellevue, WA 98004. (206) 454-1315.

MICROSOFT

MTU

CODOS

Photo credit: GREAT GALAXY IN ANDROMEDA: Palomar Observatory, California Institute of Technology

THE MTU FLOPPY DISK CONTROLLER WITH 16K RAM GIVES YOUR AIM-65 ION DRIVE POWER!

HARDWARE

- 16K 2 PORT RAM ONBOARD WITH WRITE PROTECT
- USES THE NEC-765 DISK CONTROLLER CHIP
- ROM BOOTSTRAP LOADER SPEEDS LOADING
- DMA OPERATION ALLOWS INTERRUPTS
- SUPPORTS 8 INCH DRIVES 1 OR 2 SIDED
- MAXIMUM STORAGE IS 4 MEGABYTES
- ANALOG PLL DATA SEPERATOR

SYSTEM FEATURES

- FORMAT UTILITY LOGS OUT DEFECTIVE SECTORS
- DISK/FILE COPY WITH WILDCARD SELECTION
- SYSTEM CUSTOMIZATION UTILITY
- VISIBLE MEMORY TERMINAL DRIVER PROVIDED
- INTERCHANGE CODOS SOFTWARE AMONG KIM, SYM, AIM, PET SYSTEMS
- IN FIELD USE FOR OVER 6 MONTHS

CODOS SOFTWARE

- CODOS DISK OPERATING SOFTWARE
- 8K RAM RESIDENT ALLOWS UPGRADES
- FINDS AND LOADS 32K BYTES IN 3 SECONDS
- STARTUP FILE EXECUTES AT BOOT-UP
- COMMAND FILE EXECUTION FROM DISK
- DYNAMIC DISK STORAGE ALLOCATION
- DEVICE-INDEPENDENT I/O
- TRUE RANDOM ACCESS TO RECORD IN ONE ACCESS
- MONITOR WITH 29 BUILT-IN COMMANDS
- FULL ENGLISH ERROR MESSAGES
- FILE NAMES 12 CHARACTERS + EXTENSIONS
- FILE SIZE UP TO 1 MEGABYTE
- UP TO 247 FILES PER DISK DRIVE
- INDIVIDUAL WRITE PROTECT ON FILES
- WORKS WITH AIM EDITOR, ASSEMBLER, BASIC AND MONITOR ROMS
- SUPERVISOR CALLS AVAILABLE TO USER PROGRAM

K-1013M Hardware Manual-$10, K-1013-3M CODOS manual-$25, K-1013-3D RAM/Disk controller with CODOS-$595, Floppy drives, cables, power supply also available.

MASTERCARD & VISA accepted

WRITE OR CALL TODAY FOR OUR 48 PAGE FALL 1980 CATALOG DESCRIBING ALL MTU 6502 PRODUCTS, INCLUDING 320 BY 200 GRAPHICS, AIM GRAPHIC/TEXT PRINT SOFTWARE, BANK-SWITCHABLE RAM/ROM/I-O, AIM CARD FILE, POWER SUPPLY AND MORE!

Micro Technology Unlimited • 2806 Hillsborough St. • P.O. Box 12106 • Raleigh, N.C. 27605 • (919) 833-1458

NO. 33 FEBRUARY 1981

MICRO™

THE 6502 JOURNAL

CONTENTS

DEPARTMENTS

MICRO™ is published monthly by:
MICRO INK, Inc., Chelmsford, MA 01824
Second Class postage paid at:
Chelmsford, MA 01824 and additional offices
Publication Number: COTR 395770
ISSN: 0271-9002

Subscription Rates:	*Per Year*
U.S.	
through March 31	$15.00
thereafter	$18.00
Foreign surface mail	$21.00
Air mail:	
Europe	$36.00
Mexico, Central America	$39.00
Middle East, North Africa	$42.00
South America, Central Africa	$51.00
South Africa, Far East, Australasia	$60.00

For back issues, subscriptions, change of address or other information, write to:
MICRO
P.O. Box 6502
Chelmsford, MA 01824
or call
617/256-5515

MICRO Editorial

Too Many Apples!

MICRO was founded in 1977 to provide coverage for the *entire* spectrum of 6502 microprocessor-based systems. At the time, there were only three major systems available: the KIM-1 which had been around for a year or two and had generated much of the initial interest in the 6502 microprocessor; the PET which was just starting to come off the production line in limited numbers; and, the Apple II which was also just starting to arrive in computer stores. Since then, several new systems have been added: a number of systems from Ohio Scientific, the AIM 65, the SYM-1, the Atari.

MICRO has endeavored to provide coverage of all these systems, as well as to provide 6502-related information which was not specific to any microcomputer. For reasons difficult to understand, we have received far more high-quality material about the Apple II than about the other microcomputers! This has lead to uneven coverage and other problems.

If we selected articles purely on merit, the magazine would quickly become overweighted with Apple material, because the quantity of good Apple articles would crowd out the smaller quantity of good articles on other microcomputers. A second option is no better. As we expand the number of pages in MICRO, even if we held constant the amount of space devoted to non-Apple microcomputers and assigned the additional pages to Apple material, the magazine would still be out of balance, and many non-Apple readers would feel slighted.

Furthermore, due to the quantity of good articles, the delay between submission and publication of Apple articles has grown to an unacceptable level. I feel that articles should be published as quickly as possible so that the author gets his ideas into print in a short time, so that the author can promptly collect payment for his work, and so that the reader gets up-to-date information. To solve these problems, several ideas are under consideration:

0. No change—keep on as is.

1. Print the *best* material in MICRO—without regard to which microcomputer it pertains. If this means that the Apple overwhelms the rest, so be it.

2. Allocate a larger portion of MICRO to coverage of the Apple, perhaps even adding as many as 16 to 32 extra pages, so that coverage of the other microcomputers does not diminish.

3. Publish an Apple Supplement or Apple Quarterly which would permit the Apple-specific material, which would normally not appear due to lack of space, to be printed in a timely fashion.

4. Print the "extra" Apple material in book form. This is being done to some extent with the publication of our MICRO/Apple which is scheduled to appear in April 1981. While this volume will primarily consist of material previously printed in MICRO, with numerous updates by the original authors, new listings, and optional disks with programs, other volumes could include original material which has not appeared in MICRO and which may never appear in the magazine.

5. If enough Apple material is available, publish a *monthly* Apple magazine of the calibre of MICRO (Note: There are already some 30 periodicals of various quality and frequency devoted exclusively to the Apple). At the same time, divest MICRO of all specific Apple articles and transform it to a magazine devoted to the "other" microcomputers. MICRO could still remain an overall 6502 resource publication carrying general material applicable to the Apple as well as other 6502-based computers.

Since any decision we make could greatly affect our coverage of the Apple, we would like to get some feedback from Apple readers. Which of the above options do you prefer, or can you suggest any alternatives?

Robert M. Tripp

Editor/Publisher

About the Cover

Travel Options

Travel options were few in the days of George Washington and Abraham Lincoln (whose birthdays are celebrated this month), two of the U.S. presidents portrayed here in Mt. Rushmore National Monument, South Dakota. Washington rode his horse to the capital and Lincoln took the train.

Today, when we make travel plans, we have innumerable options. There are countless, massive data bases compiled by airlines, hotels, travel agencies, camping organizations, auto clubs, and local and national tourism promotion bureaus. A few of these data bases are already directly linked to microcomputer networks such as The Source and Micronet.

In a short while, we can expect microcomputers to weigh our individual vacation whims against our budgets, present us in advance with pictures of our destination, and plot our unique travel course, with every big and little detour we have time and money for.

Designing possible trips will be fun and, for armchair travelers, even a game. (Photo by Gary Fish.)

MICRO Letterbox

Dear Editor:

I am a long-time MICRO reader and fan. While I find the magazine very informative in terms of programs and routines, I feel that there is a lack in terms of reader communications. Perhaps the readers are to blame, but I can't really believe that you receive too few letters to support a Letterbox column.

Perhaps it is editorial policy. While the magazine is generally well-filled with articles, it seems no space is set aside on a consistent basis for feedback and exchange of ideas by the readers. In issue 23 you said you hoped to make the Letterbox a monthly feature. Since then, Letterbox has only appeared in issues 26, 27, and 29. Issue 26 Letterbox was devoted to specifics on 16 bit 650x wishlists. This doesn't seem to leave much room for other general reader communications.

I suggest that you establish Letterbox permanently for 6 months. If you receive no interesting letters for the month, simply take one sentence to say so. I believe if you make the space available, readers will respond.

Frank Lawyer
126 Demott Lane
Somerset, New Jersey 08873

We are now receiving enough letters of general interest to make Letterbox a permanent department of MICRO. Despite this increase in mail, we encourage more readers to write in. Take advantage of the opportunity provided by Letterbox. Share brief communications with your fellow readers. Use Letterbox as your forum!

The Editor

Dear Editor:

I am interested in contacting people who have used their Apple computer as a text editor/word processor and have substantial text files of articles or other ordinary writing. I want to borrow the files and tear them apart into words for a research project.

For a possible commercial use, I would like to hear from people who have developed data bases on the Apple. For example, an address file that could not otherwise be sold might yield street names for a particular city for use in a game, in research, or for a developer looking for new names.

Mike Firth
104 N. St. Mary
Dallas, Texas 75214

Dear Editor:

I am interested in contacting other users of the CGRS Microtech disk operating system, CRS/DOS, which is installed on my 6502/S100 homebrew development system. I am an experienced hardware designer and novice assembly language programmer who is having trouble understanding the use of the CRS assembler, editor and certain DOS functions. Microtech appears to be growing rapidly and while the DOS appears quite powerful, they have not found time to hold my hand while I learn to use it. I'm sure that input from other users would save me a great deal of trial and error, and am willing to either pay someone or trade hardware expertise for help. Specifically, I need hints as to the meaning of disk error messages (hex), an editor manual, an assembler error message table, and a way to copy individual system routines from one diskette to another without reinitializing the entire diskette.

I am also interested in hearing from folks in the Bay Area who might wish to assist with applications programming on our real-time music performance system based on the above DOS and extensive hardware, either for cash or in exchange for hardware design and construction.

Clance
Power Bus Garage
2000 Center St. #6502
Berkeley, California 94704
(415) 549-0541 (days)
524-9586 (evenings)

Dear Editor:

I eagerly read Robert Phillips' article on "The Binary Sort" in the February 1980 issue of MICRO (21:15), although I had to work my way through the special character omissions. Mr. Phillips' approach is very reasonable but not the one that came to my mind when I recently had the same goal.

I use 3 pointers instead of 2, keeping track of the beginning (N0) and end (N1) of the range yet to be searched. The array element in the middle of this range (L1$(PT)) is compared with the item searched for (SW$), and either N1 or N0 is set to PT if the comparison fails, depending upon whether the comparison value is above or below the array element, respectively. This approach eliminates the need to check the last element of the array at the end of the search.

I have written this procedure as a subroutine, using the same variable names as Mr. Phillips where possible. I have added a test for an empty array and the variable FO which is 0 if the search has failed, or 1 if it has succeeded. If the search fails, SW$ can be inserted at position PT by first moving elements PT to TL down one and increasing TL by one.

```
 10 IF TL = 0 THEN PT = 1:
    FO = 0: RETURN
 20 FO = 1: N0 = 0: N1 = TL
    + 1
 30 PT = INT((N1 + N0) / 2)
 40 IF PT = 0 OR PT = N0
    GOTO 80
 50 IF L1$(PT) = SW$ THEN
    RETURN (found it)
 60 IF L1$(PT) > SW$ THEN
    N1 = PT: GOTO 30
 70 N0 = PT: GOTO 30
 80 PT = PT + 1
 90 FO = 0
100 RETURN (search has failed)
```


James A. Petrich, Ph.D.
5123 Sirretta
San Antonio, Texas 78233

A Simple Securities Manager for the Apple

Manage your stocks more carefully in these volatile times! Use this simple program to record security transactions, keep track of gains and losses, and evaluate your holdings at any time.

Ronald A. Guest
12153 Melody Dr. #204
Denver, Colorado 80234

```
          ALL/NOTSOLD/SOLD A
     PRESS 'RETURN' WHEN READY
NAME  PDATE  SDATE   PPRICE SPRICE  DIV

GETRI 021379 082779  1517.3   875.5   0
 200

MBI   060179         2832.3  5124.3 3.5
 100

PLUMM 031479 071579  5786.8  8514.1   0
 200

TURKE 052278          827.3  1159.5  .8
 400

4M    120579          879.3   945.8 1.3
 150


TOTALS
         PPRICES 11842.75
         SPRICES 16619.125
    PRESS 'RETURN' WHEN READY
```

Listing 1

```
CURRENT DATE (MMDDYY) 033180
            ALL/NOTSOLD/SOLD A
    PRESS 'RETURN' WHEN READY
   NAME      $GAIN   %GAIN

GETRICHQUI  -641.75   -42.3
MBI            2292   80.93
PLUMMET     2727.38   47.13
TURKEY       332.25   40.16
4M             66.5    7.56

TOTALS    $GAIN 4776.38
          %GAIN 40

    PRESS 'RETURN' WHEN READY
```

Listing 2

```
CURRENT DATE (MMDDYY) 033180
            ALL/NOTSOLD/SOLD A
    PRESS 'RETURN' WHEN READY
   NAME      $GAIN   %GAIN

MBI          258.75      9
TURKEY          550     66
4M            46.88      5

TOTALS    $GAIN 855.63
          %GAIN 19

    PRESS 'RETURN' WHEN READY
```

Listing 3

One of the many uses of a home computer is for record keeping. And one of the (hopefully) most profitable types of record to keep is security transactions. In the highly volatile economic circumstances which now exist, it has become increasingly more important to have accurate information readily at hand. In this area, a small computer can be a big help.

I have written a program to assist in making decisions about my holdings. This program runs on a 32K Apple with ROM Applesoft and a Disk II. The output of the program is heavily oriented toward the standard 24 × 40 Apple display, but as you will see, it produces adequate results when used with a hard-copy printer. Three types of reports may be generated, and four types of operations may be performed on the securities data.

The stock manager program is tailored to fit my own needs, and others may require different reports or formats. I will try to provide sufficient information in this article to allow the program to be easily modified.

Reports

The three types of reports which may be requested are: a listing of the data in the current portfolio, a listing of the appreciation in the portfolio, and a (very) rough estimate of the dividends paid by the portfolio. In all three of the reports, the user may select that all securities be listed, that all unsold securities be listed, or that all sold securities be listed.

The LIst report outputs all of the information stored in the disk file for the selected class of holdings. The information printed includes the first five characters of the name, the purchase and sale dates, the purchase and sale prices, the per share dividend, and the number of shares (listing 1). Up to five holdings may be printed per page, and the totals of the purchase prices and sale prices will be printed on the final page. For an explanation of the meaning of the sale date and sale price for a security which has not yet been sold, see the paragraphs on adding an entry and on reading a data file.

The appreciation report lists the dollar and percent gains (losses) for each of the stocks listed. At the end of the report, the total dollar gain and the percent gain (loss) based on the purchase price are printed for the holdings selected (listing 2). If a security was sold 12 or more months after it was purchased, or if the security was purchased 12 or more months prior to the current date, then the name is displayed in inverse video indicating that the holding may be eligible for long-term gain. Since the printer will not output inverse characters, a box was drawn around the names of stocks falling into this category.

A report of the dividends paid for the selected stocks provides an estimate of the dollar amount paid from the time the security was purchased to the time it was sold (or the current date if not yet sold). Only the selected securities with non-zero dividends are listed. The estimate is based on the number of months a security was held (listing 3). Since most securities pay dividends on specific dates, holdings which are quickly sold may show a dividend on the report, but have never been paid out. Since my investment goals are heavily oriented toward capital appreciation, the discrepancy does not bother me. People with different investment goals may wish to improve the estimates.

Operations on Data

The stock manager stores information in a sequential text file. A free format is used which allows each element to vary in length. The first element of the data file is a count of the number of entries in that file. The remainder of the file contains the entries. A security's entry, in the order of appearance, is: name, purchase date, sale date, purchase price, sale price, dividend, and number of shares.

When first run, the stock manager will have no entries, so the first command to execute is the ADd command. ADd requests the information which will be stored in the data file. All dates should be entered in the form MMDDYY with no slashes or other separators. The date must be six characters in length, so each field must be zero-filled. For instance, February 2, 1979 would be entered as 020279. When adding an entry for an as yet unsold security, enter a single blank for the sale date.

After adding all of the entries desired, a WRite command should be performed. WRite will prompt for a file name, and then output the entries to disk. Before any reports are generated, a REad command should be executed. The REad will ask for the file name and then read the data file. After closing the data file, REad will prompt for the current price of all holdings which have not yet been sold. This price is then used in generating reports. Note that the price entered should be the total price, not the per share price.

If an error is made adding an entry, or if a holding is sold, the data may be updated with the CHange command. CHange searches for the given name and then requests the new information. If a holding is to be deleted, enter an * for the name. Be sure to do a WRite if the changes are to be permanent. If more than one entry in a portfolio has the same name (to the 25th character), the month purchased or some other difference should be introduced to allow a unique search. When the stock manager is EXited, it asks if the file should be updated. An answer of 'yes' will cause a WRite to be performed.

The stock manager was written to allow new commands or data fields to be added easily. To add a command, choose an unused entry in CMD$ (denoted by 'XX') and substitute the first two characters of the new command (lines 130-133). Between lines 330-399, output the command name and description for the menu. On line 510, change the entry in the GOSUB list corresponding to the index into CMD$ to the line number of the new command.

Adding a new data field is just as easy. Simply dimension the new field appropriately in lines 100-110. Then add a line in 36240-36280 to input the field, add a line in 38240-38255 to print the field, and add a line in 40110-40190 to enter the field into the data area. A list of the major variables and their usage is given in table 1 and a list of the subroutines is in table 2.

***Table 1:* List of variables.**

ANS	Indicates what class of stocks to list All(0)/Notsold(1)/Sold(2)
CC	Index of last entry in CMD$
CD$	Current date
CMD$	Array of two character command names
COUNT	Number of holdings in current file
D$	Control-D for DOS
DG	Dollar gain
DV	Array of per share dividends
F$	File name containing stocks
INDEX	Index to stock holdings
LINE	Number of lines being displayed
MN	Number of months between sale (or current) date and purchase date
NM$	Array of stock names
PD$	Array of purchase dates
PP	Array of purchase prices
SD$	Array of sale dates (1 blank if not sold)
SH	Array of number of shares
SP	Array of sale prices
TPP	Total purchase prices
TSP	Total sale prices
TV	Same as TPP
YR	Number of years between sale (or current) date and purchase date

Persons without a disk may also use this program by changing the REad routine to use BASIC READ and DATA statements. The WRite, CHange, and ADd routines can then be deleted since changes to the entries can be made by retyping the appropriate DATA statement. With these modifications, the program should easily run on a 16K cassette system (Applesoft in ROM).

Listing 4

```
10   REM
        STOCK HOLDINGS MANAGER

20   REM              BY
25   REM       R. A. GUEST
30   REM
          COPYRIGHT (C) 1980 R. A.
      GUEST

100   DIM NM$(25),PD$(25),SD$(25),
      PP(25),SP(25),DV(25)
101   DIM CMD$(10),SH(25)
120   REM
          INIT COMMAND STRINGS

130 CMD$(0) = "AP":CMD$(1) = "EX"
     :CMD$(2) = "CH"
131 CMD$(3) = "XX":CMD$(4) = "DI"
     :CMD$(5) = "XX"
132 CMD$(6) = "LI":CMD$(7) = "XX"
     :CMD$(8) = "RE"
133 CMD$(9) = "WR":CMD$(10) = "AD
     "
135 COUNT = 0
140 CC = 10: REM LAST COMMAND
150 D$ =  CHR$ (4)
200   TEXT : HOME
210   VTAB 8: HTAB 12
220   PRINT "STOCK MANAGER 1.0"
230   VTAB 12: HTAB 13: INVERSE
240   PRINT "BY R. A. GUEST": NORMAL

250   FOR I = 1 TO 1000: NEXT I
300   REM DISPLAY MENU
310   HOME :T =  FRE (0): REM CLEA
      N UP STRINGS
320   VTAB 2: HTAB 18
325   REM
          PRINT COMMANDS

330   PRINT "MENU"
340   VTAB 4: INVERSE : PRINT "ADD
      ";: NORMAL : PRINT " HOLDING
      "
350   INVERSE : PRINT "APPRECIATIO
      N"
360   PRINT "CHANGE";: NORMAL : PRINT
       " HOLDING"
370   INVERSE : PRINT "DIVIDENDS":
       NORMAL
380   INVERSE : PRINT "LIST";: NORMAL
      : PRINT " HOLDINGS"
390   INVERSE : PRINT "READ";: NORMAL
      : PRINT " DATA FILE"
395   INVERSE : PRINT "WRITE";: NORMAL
      : PRINT " DATA FILE"
399   INVERSE : PRINT "EXIT": NORMAL

400   VTAB 22: HTAB 10
410   INPUT "COMMAND: ";YN$
415   REM
          SEARCH FOR COMMAND

420   FOR I = 0 TO CC: IF CMD$(I) =
       LEFT$ (YN$,2) GOTO 500
430   NEXT
440   GOTO 400
500 I = I + 1
510   ON I GOSUB 20000,18000,24000
      ,19000,28000,19000,32000,190
      00,36000,38000,40000
600   GOTO 300
18000   REM
          EXIT

18020   INPUT "DO YOU NEED TO UPDA
     TE FILE ";YN$
18040   IF  LEFT$ (YN$,1) = "Y" THEN
      GOSUB 38000: REM
     CLEAR AND UPDATE
18060   END
19000   REM
          UNIMPLEMENTED

19040   PRINT "NO SUCH COMMAND"
19060   RETURN
20000   REM
          CAPITAL GAINS(AP)

20010   REM
          HOLDINGS >1 YEAR

20020   REM
          INVERSED FOR LTG

20080   INPUT "CURRENT DATE (MMDDY
     Y) ";CD$
20100   HOME : VTAB 10: HTAB 13
20120   INPUT "ALL/NOTSOLD/SOLD ";
     YN$
20140 ANS = 0: IF  LEFT$ (YN$,1) =
     "N" THEN ANS = 1
20160   IF  LEFT$ (YN$,1) = "S" THEN
     ANS = 2
20200   REM
20210 INDEX = 0: HOME :LINE = 30:
     DG = 0:TV = 0
20220   IF INDEX >  = COUNT GOTO 2
     0900: REM DONE
20230   IF ANS = 0 GOTO 20300
20240   IF (ANS = 1) AND (SD$(INDE
     X) <  > " ") GOTO 20540
20260   REM
          USE 'ADD' TO ENTER INF
     OR
```

(continued)

Listing 4 (continued)

```
20300  REM OUTPUT HEADER
20320  IF LINE > 18 THEN  GOSUB 5
    2000
20330 F1 = 0: REM IF NOT SOLD, US
    E CURRENT DATE
20340  IF SD$(INDEX) = " " THEN F
    1 = 1:SD$(INDEX) = CD$
20349  REM
        CALCULATE YEAR DIFFERE
    NCE

20350 TP =  VAL ( RIGHT$ (SD$(IND

    EX),2)) -  VAL ( RIGHT$ (PD$
    (INDEX),2))
20351 TP = TP * 12: REM CONVERT T
    O MONTHS
20355  REM
        CALCULATE MONTH DIFFER
    ENCE

20360 TP = TP +  VAL ( LEFT$ (SD$
    (INDEX),2)) -  VAL ( LEFT$ (
    PD$(INDEX),2))
20362  REM
        DELETE ENTRY

20365  IF TP < 12 GOTO 20395
20370  INVERSE : REM LONG TERM GA
    IN
20395  IF F1 THEN SD$(INDEX) = "
    "
20400  PRINT  LEFT$ (NM$(INDEX),1
    0);: NORMAL : HTAB 12
20410  REM
        CALCULATE DOLLAR GAIN

20420 TP$ =  STR$ ( INT ((SP(INDE
    X) - PP(INDEX)) * 100 + .5) /
    100)
20430  IF  LEN (TP$) < 8 THEN TP$
     = " " + TP$: GOTO 20430
20440  PRINT TP$;: HTAB 20
20450 DG = DG +  VAL (TP$): REM T
    OTAL DOLLAR VALUE
20460 TV = PP(INDEX) + TV: REM TO
    TAL VALUE
20465  REM
        CALCULATE % GAIN

20470 TT = ( VAL (TP$) / PP(INDEX
    )) * 100
20480 TT$ =  STR$ ( INT (TT * 100
     + .5) / 100): REM PERCENT G
    AIN
20490  IF  LEN (TT$) < 7 THEN TT$
     = " " + TT$: GOTO 20490
20500  PRINT TT$
20520 LINE = LINE + 1
20540 INDEX = INDEX + 1
20560  GOTO 20220: REM  DO NEXT O
    NE
20890  REM
        PRINT TOTALS

20900  PRINT : PRINT "TOTALS";: HTAB
    10: PRINT "$GAIN ";DG
20910  IF TV = 0 GOTO 20940
20920  HTAB 10: PRINT "%GAIN ";( INT
    ((DG / TV) * 100 + .5))
20940  PRINT
20960  GOSUB 51000: REM WAIT FOR
    KEY PRESS
20970  RETURN
24000  REM
        CHANGE/DELETE HOLDING

24020  REM
        INPUT '*' FOR NAME TO

    DELETE

24040  REM
        INPUT A BLANK FOR SALE
     DATE IF NOT YET SOLD

24200  INPUT "SEARCH STRING ";TS$

24220  FOR K = 0 TO (COUNT - 1)
24222  IF TS$ =  LEFT$ (NM$(K), LEN
    (TS$)) GOTO 24300
24225  NEXT K
24240  PRINT "NOT FOUND": FOR KK =
    1 TO 300: NEXT : RETURN
24300 TP = COUNT:COUNT = K
24302  PRINT NM$(K): PRINT PD$(K)
    : PRINT SD$(K): PRINT PP(K):
     PRINT SP(K): PRINT DV(K): PRINT
    SH(K)
24320  PRINT "ENTER '*' FOR NAME
    TO DELETE."
24330  FOR KK = 1 TO 400: NEXT
24340  GOSUB 40100: REM GET FIELD
    S
24360  IF NM$(K) <  > "*" THEN CO
    UNT = TP: RETURN
24365 COUNT = COUNT - 1
24367  REM
        MOVE REST DOWN IN LIST


24370  FOR K = COUNT TO TP - 2
24380 K1 = K + 1
24390 NM$(K) = NM$(K1):PD$(K) = P
    D$(K1):SD$(K) = SD$(K1)
24400 PP(K) = PP(K1):SP(K) = PP(K
    1):DV(K) = DV(K1):SH(K) = SH
    (K1)
24420  NEXT
24440 COUNT = TP - 1
24460  RETURN
```

```
26000  REM
          CLEAR SALE PRICE OF UN
     SOLDS

26100  FOR I = 0 TO COUNT - 1
26120  IF SD$(I) = " " THEN SP(I)
     = 0
26140  NEXT
26200  RETURN
28000  REM
          ESTIMATE DIVIDEND GAIN

28020  INPUT "CURRENT DATE (MMDDY
     Y) ";CD$
28040  HOME : VTAB 10: HTAB 13
28060  INPUT "ALL/NOTSOLD/SOLD ";
     YN$
28080 ANS = 0: IF  LEFT$ (YN$,1) =
     "N" THEN ANS = 1
28100  IF  LEFT$ (YN$,1) = "S" THEN
     ANS = 2
28120 INDEX = 0: HOME :LINE = 30:
     DG = 0:TV = 0
28180  REM TEST IF DONE
28200  IF INDEX >  = COUNT THEN 2

     8900
28220  IF ANS = 0 GOTO 28280
28240  IF (ANS = 1) AND (SD$(INDE
     X) <  > " ") GOTO 28620
28260  IF (ANS = 2) AND (SD$(INDE
     X) = " ") GOTO 28620
28270  REM PRINT HEADER
28280  IF LINE > 18 THEN  GOSUB 5
     2000
28290  REM
          USE CURRENT DATE OR UN
     SOLDS

28300  IF DV(INDEX) = 0 GOTO 2862
     0: REM DON'T USE
28305 F1 = 1
28310  IF SD$(INDEX) = " " THEN F
     1 = 1:SD$(INDEX) = CD$
28315  REM
          CALCULATE MONTHS

28320 MN =  VAL ( LEFT$ (SD$(INDE
     X),2)) -  VAL ( LEFT$ (PD$(I
     NDEX),2))
28323  REM
          CALCULATE YEARS

28325 YR =  VAL ( RIGHT$ (SD$(IND
     EX),2)) -  VAL ( RIGHT$ (PD$
     (INDEX),2))
28327  REM
          CONVERT TO MONTHS

28330 MN = MN + YR * 12
28340  IF F1 THEN SD$(INDEX) = "
     "
28400  PRINT  LEFT$ (NM$(INDEX),1
     0);: HTAB 12
28410  REM
          ESTIMATE DIVIDENDS PAI
     D

28420 TP =  INT ((DV(INDEX) * SH(
     INDEX) * (MN / 12)) * 100 +
     .5) / 100
28440 TP$ =  STR$ (TP)
28460  IF  LEN (TP$) < 8 THEN TP$
     = " " + TP$: GOTO 28460
28480  PRINT TP$;: HTAB 20
28490  REM
          CALCULATE DOLLAR GAIN
     AND
28495  REM
          TOTAL VALUE

28500 DG = DG +  VAL (TP$):TV = T
     V + PP(INDEX)
28510  REM
          CALCULATE % GAIN

28520 TT =  INT (( VAL (TP$) / PP
     (INDEX)) * 100 + .5)
28540 TT$ =  STR$ (TT)
28560  IF  LEN (TT$) < 7 THEN TT$
     = " " + TT$: GOTO 28560
28580  PRINT TT$
28600 LINE = LINE + 1
28620 INDEX = INDEX + 1
28640  GOTO 28200
28900  GOSUB 20900: REM
                        OUTPUT TO
     TALS

28920  RETURN
32000  REM
          LIST CURRENT HOLDINGS

32100  HOME : VTAB 10: HTAB 10
32110  INPUT "ALL/NOTSOLD/SOLD ";
     YN$
32120 ANS = 0: REM ALL
32130  IF  LEFT$ (YN$,1) = "N" THEN
     ANS = 1: REM   NOTSOLD
32140  IF  LEFT$ (YN$,1) = "S" THEN
     ANS = 2: REM SOLD
32210 INDEX = 0: HOME :LINE = 30:
     TPP = 0:TSP = 0
32300  IF INDEX >  = COUNT GOTO 3
     2900
32302  IF ANS = 0 GOTO 32310
32304  IF (ANS = 1) AND (SD$(INDE
     X) = " ") GOTO 32310
32306  IF (ANS = 2) AND (SD$(INDE
     X) <  > " ") GOTO 32310
32308 INDEX = INDEX + 1: GOTO 323
     00
```

(continued)

Listing 4 (continued)

```
32310  IF LINE > 18 THEN  GOSUB 5
     0000: REM WAIT AND PRINT HEA
     DER
32320  PRINT  LEFT$ (NM$(INDEX),5
     );: HTAB 7
32330  PRINT  LEFT$ (PD$(INDEX),6
     );: HTAB 14
32340  PRINT  LEFT$ (SD$(INDEX),6
     );: HTAB 21
32350  REM
          PURCHASE PRICE

32360 TP$ =  STR$ ( INT (PP(INDEX
     ) * 10.0 + 0.5) / 10.0)
32380  IF  LEN (TP$) < 7 THEN TP$
      = " " + TP$: GOTO 32380
32390  PRINT TP$;: HTAB 29
32395  REM
          SALE PRICE

32400 TP$ =  STR$ ( INT (SP(INDEX
     ) * 10.0 + 0.5) / 10.0)
32410  IF  LEN (TP$) < 7 THEN TP$
      = " " + TP$: GOTO 32410
32420  PRINT TP$;: HTAB 37
32425  REM
          DIVIDEND

32430 TP$ =  STR$ ( INT (DV(INDEX
     ) * 10.0 + 0.5) / 10.0)
32440  IF  LEN (TP$) < 3 THEN TP$
      = " " + TP$: GOTO 32440
32450  PRINT TP$
32455  REM
          NUMBER OF SHARES

32460  PRINT " ";SH(INDEX)
32465  REM
          COMPUTE TOTAL SALES AN
     D

32466  REM
          TOTAL PURCHASE PRICES
32470 TSP = TSP + SP(INDEX):TPP =
     TPP + PP(INDEX)
32480  PRINT
32800 LINE = LINE + 3
32810 INDEX = INDEX + 1
32820  GOTO 32300
32880  REM
          PRINT TOTALS

32900  PRINT : PRINT "TOTALS"
32910  HTAB 10: PRINT "PPRICES ";
     TPP
32920  HTAB 10: PRINT "SPRICES ";
     TSP
32960  GOSUB 51000: REM WAIT FOR
     KEY PRESS
32970  RETURN
36000  REM
          READ STOCK LISTING FIL
     E

36100  INPUT "FILE NAME ";F$
36120  PRINT D$;"OPEN ";F$
36140  PRINT D$;"READ ";F$
36200  INPUT COUNT
36220  FOR I = 0 TO (COUNT - 1)
36240  INPUT NM$(I): INPUT PD$(I)
     : INPUT SD$(I)
36260  INPUT PP(I): INPUT SP(I)
36280  INPUT DV(I): INPUT SH(I)
36285  REM
          CHECK FOR NOT SOLD

36290  IF  LEN (SD$(I)) < 6 THEN
     SD$(I) = " "
36300  NEXT
36320  PRINT D$;"CLOSE ";F$
36325  REM
          GET PRICES FOR STOCKS
     NOT SOLD

36330  FOR I = 0 TO (COUNT - 1)
36340  IF SD$(I) <  > " " GOTO 36
     370
36350  PRINT NM$(I)
36360  INPUT "CURRENT PRICE ";SP(
     I)
36370  NEXT
36400  RETURN
38000  REM
          UPDATE STOCK LISTING F
     ILE

38050  GOSUB 26000: REM CLEAR NOT
     SOLD PRICES
38100  INPUT "FILE NAME ";F$
38120  PRINT D$;"OPEN ";F$
38140  PRINT D$;"WRITE ";F$
38200  PRINT COUNT
38220  FOR I = 0 TO (COUNT - 1)
38240  PRINT NM$(I): PRINT PD$(I)
     : PRINT SD$(I)
38242  PRINT PP(I): PRINT SP(I): PRINT
     DV(I): PRINT SH(I)
38260  NEXT
38300  PRINT D$;"CLOSE ";F$
38320  RETURN
40000  REM
          ADD A HOLDING

40080  HOME : VTAB 4
40100  INPUT "NAME ";NM$(COUNT)
40110  PRINT "INPUT DATES IN THE
     FORM (MMDDYY)"
40120 NM$(COUNT) =  LEFT$ (NM$(CO
     UNT),25)
40140  INPUT "PURCH DATE ";PD$(CO
     UNT):PD$(COUNT) =  LEFT$ (PD
     $(COUNT),6)
```

```
40145  PRINT "ENTER A SINGLE BLAN
     K IF NOT SOLD"
40150  INPUT "SALE DATE ";SD$(COU
     NT):SD$(COUNT) =  LEFT$ (SD$
     (COUNT),6)
40160  INPUT "PURCH PRICE ";PP(CO
     UNT)
40170  INPUT "SALE PRICE ";SP(COU
     NT)
40180  INPUT "DIVIDEND/SHARE ";DV
     (COUNT)
40190  INPUT "SHARES ";SH(COUNT)
40300 COUNT = COUNT + 1
40400  RETURN
50000  REM
           WAIT FOR (CR) THEN
50010  REM
           OUTPUT HEADING FOR 'LI
     ST'
50020  REM
50100  GOSUB 51000: HOME
50110  PRINT "NAME  ";
50120  PRINT "PDATE  ";
50130  PRINT "SDATE   ";
50140  PRINT "PPRICE ";
50150  PRINT "SPRICE  ";
50160  PRINT "DIV "
50170  PRINT
50200 LINE = 2
50300  RETURN
51000  REM
           WAIT FOR (CR) TO BE PR
     ESSED

51010  VTAB 23: HTAB 5
51020  PRINT "PRESS 'RETURN' WHEN
      READY "
51050  POKE  - 16368,0
51100  IF  PEEK ( - 16384) = 141 THEN
      RETURN
51200  GOTO 51100
52000  REM
           WAIT FOR (CR) AND

52020  REM
           PRINT HEADER

52040  REM
           FOR APPRECIATION AND D
     IVIDEND

52060  GOSUB 51000: HOME : HTAB 4

52080  PRINT "NAME";: HTAB 14
52100  PRINT "$GAIN";: HTAB 21
52120  PRINT "%GAIN"
52140  PRINT
52160 LINE = 2
52180  RETURN
```

A complete listing of the program is given in listing 4, and a sample of the displayed menu is given in listing 5. The program is fairly well documented, and I hope that the comments, along with the text of this article will enhance the usefulness of the program.

***Table 2:* Routines and their uses**

20000-21999	Appreciation Report
24000-25999	Change an Entry
28000-29999	Estimated Dividends Report
32000-33999	List Securities Entries
36000-37999	Read Securities from Disk
38000-39999	Write Securities to Disk
40000-41999	Add a New Entry
50000-50500	Print Header for List of Securities
51000-51500	Wait for Return to be Pressed
52000-52500	Print Header for Appreciation and dividend

Listing 5

```
        STOCK MANAGER 1.0
         BY R. A. GUEST
              MENU

ADD HOLDING
APPRECIATION
CHANGE HOLDING
DIVIDENDS
LIST HOLDINGS
READ DATA FILE
WRITE DATA FILE
EXIT
          COMMAND: RE
FILE NAME STOCKS EXAMPLE
MBI
CURRENT PRICE 5124.25
TURKEY
CURRENT PRICE 1159.50
4M
CURRENT PRICE 945.75
                    MENU
ADD HOLDING
APPRECIATION
CHANGE HOLDING
DIVIDENDS
LIST HOLDINGS
READ DATA FILE
WRITE DATA FILE
EXIT
          COMMAND:
```

MICRO

OSI

SOFTWARE FOR OSI

OSI

A JOURNAL FOR OSI USERS!!

The Aardvark Journal is a bimonthly tutorial for OSI users. It features programs customized for OSI and has run articles like these:

1) Using String Variables.

2) High Speed Basic On An OSI.

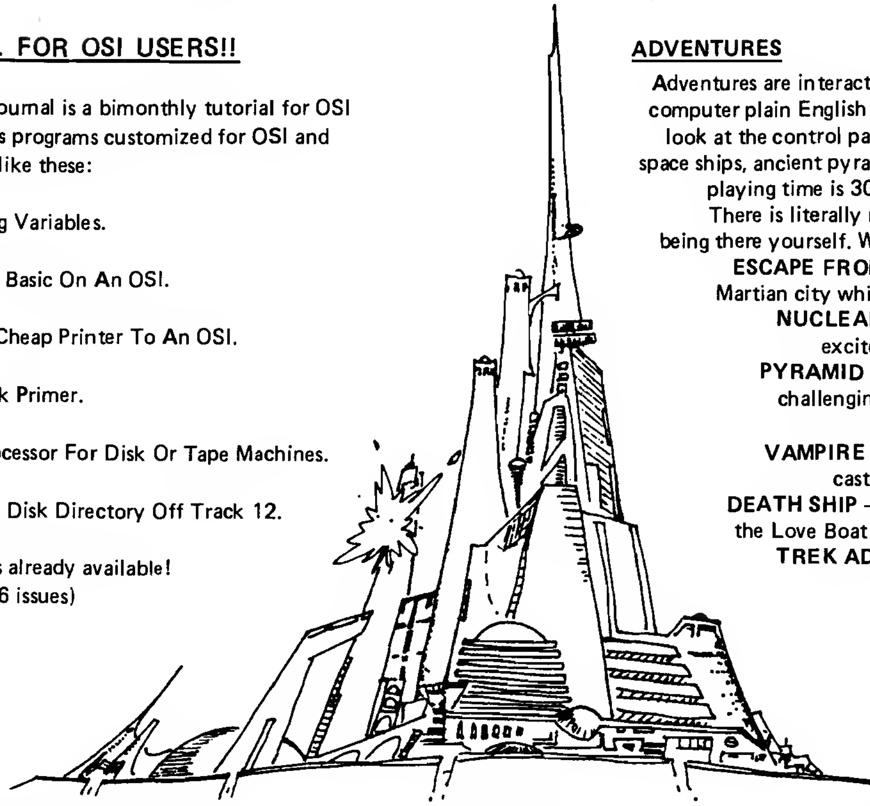
3) Hooking a Cheap Printer To An OSI.

4) An OSI Disk Primer.

5) A Word Processor For Disk Or Tape Machines.

6) Moving The Disk Directory Off Track 12.

Four back issues already available!
$9.00 per year (6 issues)



ADVENTURES

Adventures are interactive fantasies where you give the computer plain English commands (i.e. take the sword, look at the control panel.) as you explore alien cities, space ships, ancient pyramids and sunken subs. Average playing time is 30 to 40 hours in several sessions. There is literally nothing else like them — except being there yourself. We have six adventures available.

ESCAPE FROM MARS — Explore an ancient Martian city while you prepare for your escape.

NUCLEAR SUBMARINE — Fast moving excitement at the bottom of the sea.

PYRAMID — Our most advanced and most challenging adventure. Takes place in our own special ancient pyramid.

VAMPIRE CASTLE — A day in old Drac's castle. But it's getting dark outside.

DEATH SHIP — It's a cruise ship — but it ain't the Love Boat and survival is far from certain.

TREK ADVENTURE — Takes place on a familiar starship. Almost as good as being there.

$14.95 each

NEW SUPPORT ROMS FOR BASIC IN ROM MACHINES

C1S — for the C1P only, this ROM adds full screen edit functions (insert, delete, change characters in a basic line.), Software selectable scroll windows, two instent screen clears (scroll window only and full screen.), software choice of OSI or standard keyboard format, Bell support, 600 Baud cassette support, and a few other features. It plugs in in place of the OSI ROM. NOTE: this ROM also supports video conversions for 24, 32, 48, or 64 characters per line. All that and it sells for a measly $39.95.

C1E/C2E for C1/C2/C4/CB Basic in ROM machines.
This ROM adds full screen editing, software selectable scroll windows, keyboard correction (software selectable), and contains an extended machine code monitor. It has breakpoint utilities, machine code load end save, block memory move and hex dump utilities. A must for the machine code programmer replaces OSI support ROM. Specify system $59.95

DISK UTILITIES

SUPER COPY — Single Disk Copier
This copy program makes multiple copies, copies track zero, and copies ell the tracks that your memory can hold at one time — up to 12 tracks at a pass. It's almost as fast as dual disk copying. — $15.95

MAXIPROSS (WORD PROCESSOR) — 65D polled keyboard only - has globel and line edit, right and left margin justification, imbedded margin commands, choice of single, double or triple spacing, file access capabilities and all the features of a major word processor — and it's only $39.95.

P.C. BOARDS

MEMORY BOARDS!! — for the C1P. — end they contain parallel ports!
Aardvarks new memory board supports 8K of 2114's and has provision for e PIA to give a parallel ports! It sells as a bare board for $29.95. When assembled, the board plugs into the expansion connector on the 600 board. Aveileble now!

PROM BURNER FOR THE C1P — Burns single supply 2716's. Bare board — $24.95.

MOTHER BOARD — Expand your expansion connector from one to five connectors or use it to adapt our C1P boards to your C4/8P. - $14.95.

ARCADE AND VIDEO GAMES

ALIEN INVADERS with machine code moves — for fast action. This is our best invaders yet. The disk version is so fast that we had to add selectable speeds to make it playable.
Tape - $10.95 — Disk - $12.95

TIME TREK (8K) — real time Startrek action. See your torpedoes move across the screen! Reel graphics — no more scrolling displays. $9.95

STARFIGHTER — a real time space war where you face cruisers, bettleships end fighters using a variety of weepons. Your screen conteins working instrumentation and a reel time display of the alien ships. $6.95 in black and white - $7.95 in color end sound.

MINOS — A game with amazing 3D graphics. You see a maze from the top, the screen blanks, and than you are in the maze at ground level, finding your way through on foot. Realistic anough to cause claustrophobie. — $12.95

SCREEN EDITORS

These progrems all allow the editing of basic lines. All assume that you are using the standard OSI video display and polled keyboard.

C1P CURSOR CONTROL — A program that uses no RAM normally available to the system. (We hid it in unused space on page 2). It provides real backspace, insert, delete and replace functions and an optional instant screen clear.
$11.95

C2/4 CURSOR. This one uses 366 BYTES of RAM to provide a full screen editor. Edit and change lines on any part of the screen. (Basic in ROM systems only.)

FOR DISK SYSTEMS — (65D, polled keyboard and standard video only.)

SUPERDISK. Contains a basic text editor with functions similar to the above programs and also contains a renumberer, variable table maker, search and new BEXEC* programs. The BEXEC* provides a directory, create, delete, and change utilities on one track and is worth having by itself. — $24.95 on 5" disk - $26.95 on 8".

AARDVARK IS NOW AN OSI DEALER!

Now you can buy from people who can support your machine.

—THIS MONTH'S SPECIALS—

Superboard II	$279
C1P Model II	429
C4P	749

. . . and we'll include a free Text Editor Tape with aach machine!

Video Modification Plans and P.C. Boards for C1P as low as $4.95

This is only a partial listing of what we have to offer. We now offer over 100 programs, data sheets, ROMS, and boards for OSI systems. Our $1.00 catalog lists it all and contains free program listings and programming hints to boot.

OSI **Aardvark Technical Services • 1690 Bolton • Walled Lake, MI 48088** OSI
(313) 699-3110 or (313) 624-6316

Why WAIT?

The WAIT function in the OSI and PET BASICs were intended for Input/Ouput use but can have other interesting applications if you understand how to use it.

Robert L. Elm
446 Rothbury Ave.
Bolingbrook, Illinois 60439

When I first saw the WAIT function I immediately dismissed it as something I had no use for. I don't have a printer and I'm not planning on attaching my C1P to anything else, so why would I ever need WAIT? The answer lies in understanding how the function works. We seldom develop applications for things we don't understand, so what we need is the necessary foundation.

Ohio Scientific's *BASIC in ROM Reference Manual* states that "WAIT I,J,K reads status of memory location I (Decimal) exclusive OR's it with K, then AND's the result with J until a non zero result is obtained. If K is omitted it is zero." That's very nice and to the point but it doesn't tell you anything about how to determine what values should be in J and K. If I want to check a flag word or wait for some other indicator before continuing the program, I may have to detect my indicator in the presence of other changing bits. How can I determine what values should be in WAIT to do that?

The key is in knowing what the exclusive OR can do for you as a programmer. Dr. DeJong recently gave an explanation of each logical function (see MICRO 22:31) but only briefly mentioned the one application we need here. Let's suppose we have an address containing a binary word 11111110 (D.254) and we want to detect if bit 5 changes regardless of what else happens. If only bit 5 changes we would expect a binary 11011110 (D.222), but we can't guarantee only that unique value. Wouldn't it be nice if we could "blind" the machine to the normal contents of the address and then indicate only which bit we are interested in? This is exactly what we can do with the correct values in WAIT. Let's see how.

If we place the address in WAIT as "I" and its normal contents (D.254 in our example) in "K", we will in effect exclusive OR 254 with itself. Using Dr. DeJong's truth tables, what will be the result? Very interesting! If you exclusive OR any value with itself you get all zeros. In other words, the machine is now "blind" to the normal value at the address specified. Just what we wanted!

Next we want to tell it to stop "waiting" if only bit 5 changes. If we put D.32 in the WAIT instruction as "J" (2 to the 5th power is 32), it will be ANDed with the result and continue with the program only if bit 5 changes.

That's great, but is it really necessary to figure the powers of two, etc.? No, but I did want you to understand what was going on. In reality it's quicker to figure the decimal difference between the normal value and the new value we want to watch for. Using our example, 254 – 222 = 32. That's much easier, especially for figuring combinations of bits.

So what good is it if you don't have a printer? I have already covered one use in a previous article (see "A C1P Users Notebook", MICRO 31:11) to read tapes without loading them into memory. That application didn't use the "K" operator since the ACIA Status Register didn't have any extraneous data to get rid of, and I only needed to know when the Receive Data Register was full.

Recently I found another interesting use for WAIT. It seems that due to the way the circuitry is built for the polled

Table 1

Key	57088 value when depressed	wait until J	wait until K*	wait if J	wait if K
RPT	126	128	254	128	126
ESC	222	32	254	32	222
L Sh	250	4	254	4	250
R Sh	252	2	254	2	252

*Normal value of 57088 with SHIFT LOCK depressed is 254.

keyboard, if you PEEK (57088), (actually any address between 56320 and 57343 gives the same results) you will be polling Row 0. This means you don't have to disable Control C and write your own keyboard polling routine if you want a simple response from the operator. I wanted to display eight pages of information and have the computer wait between them until I was ready for the next one. The simple routine XXXX PRINT "Depress ESC to proceed": WAIT 57088,32,254: RETURN works perfectly. And now you know where I got the values for my example.

By rearranging the values it is also possible to wait only if a key is pressed. This would allow a long program such as a data listing to run continuously until you had a reason to stop it. To do this the WAIT is imbedded in the loop so that loop operation is suspended only as long as a particular key remains depressed.

A slight variation of this is to use WAIT 57088,1,255 in a loop. Now the loop can be suspended indefinitely by releasing the SHIFT LOCK key and you don't have to keep one finger on the keyboard.

At this point I must mention the uniqueness of the CTRL key. Due to its use with "CTRL C", 57088 may read either 190 or 255 when it is read. It seems to depend on the kind of instruction preceding the read. I mention this because in some cases WAIT 57088,1,255 will stop the loop forever if the CTRL key is depressed. Personally, I have decided not to use the CTRL key with WAIT so that unnecessary confusion is avoided.

Table 1 shows the WAIT 57088 J and K values for the four usable keys in Row 0. Values are given for both "wait until the key is depressed" and "wait if the key is depressed" applications. You can use any of these keys in the previously described subroutine, or use the values in IF statements to get real-time responses. The values shown assume the SHIFT LOCK key is depressed but according to my tests, it rarely affects operation either way.

With the above information you should be able to use WAIT in many different ways. In fact, grab your computer and try out one of these subroutines right now. Why wait?

MICRO

An Atari Assembler

This article describes a simple, one-pass assembler written in BASIC for a 16K Atari 400 or 800 computer system.

William L. Colsher
4328 Nutmeg Lane, Apt. 111
Lisle, Illinois 60532

Back in the first year of MICRO, our favorite editors published an article by Michael J. McCann titled "A Simple 6502 Assembler for the PET" (6:17). When I finally broke down and bought a 6502-based Atari 800 (previously I had only used 8080 and Z-80-based machines), I also picked up all the copies of MICRO that I could find. I quickly found McCann's article and decided it would be a good way to learn the Atari BASIC, and master 6502 machine language. The program that accompanies this article is (I *think*) functionally identical with the original PET version. But... I'm new at 6502 and I could easily have overlooked something.

Six Functions

The assembler presented here has six functions:

1. Input and assemble source code.
2. Save object code on tape.
3. Load object code from tape.
4. Execute the object program.
5. Call the object program as a USR routine.
6. List the object program to the screen.

The careful reader will note that these functions are nearly identical with those of the McCann assembler. Their actual use has been modified only as dictated by differences between the PET and Atari. The following paragraphs describe the use of each of the six functions.

Function 1 allows you to enter your program. Since this is a one-pass assembler you are not allowed symbolic addresses or operands (that is, labels). All addresses and operands must be entered in decimal. For example 100_{16} must be entered as 256.

In addition to the standard 6502 mnemonics, three pseudo-ops have been provided. (Pseudo-ops are instructions to the assembler that do not generate any machine code.)

> ORG—tells the assembler where to start putting your program.
>
> DC—places a number from 0 to 255 in the current location.
>
> END—tells the assembler that you are done entering code.

```
10 DIM HX$(2),SX$(1),UN$(1),MN$(1281),BY(256),
   CO$(16),T$(5),M$(5),A$(15),V$(15)
11 DIM B1$(2),B2$(2),B3$(2),AD$(4),S3$(1),
   S2$(1),S$(1),U$(1)
20 FOR E=0 TO 255
30 READ T$,T:MN$(E*5+1)=T$:BY(E)=T:T$=""
40 NEXT E
50 T$=""
60 FOR E=1 TO 16
70 READ T$:CO$(E)=T$:T$=""
80 NEXT E
90 GRAPHICS 0
100 PRINT "1. Input source code and Assemble"
110 PRINT "2. Save Object Code on Tape"
120 PRINT "3. Load Object Code from Tape"
130 PRINT "4. Execute Machine Language Program"
140 PRINT "5. Call Machine Language Program"
145 PRINT "    as USR Routine"
150 PRINT "6. List Machine Language Program"
170 INPUT T:IF (T<=0) OR (T>6) THEN GOTO 170
180 ON T GOSUB 14000,20000,9000,10000,11000,2900
190 GOTO 90
1000 SX=INT(DC/16)
1010 UN=DC-(SX*16)
1020 SX$=CO$(SX+1)
1030 UN$=CO$(UN+1)
1040 HX$(1)=SX$:HX$(2)=UN$
1050 RETURN
```

(continued)

```
2900 GRAPHICS 0
2910 PRINT "Start address";:INPUT AD:I=0
3000 IF I=23 THEN GOTO 5050
3001 I=I+1
3005 IB=PEEK(AD)
3010 T$=MN$(IB*5+1)
3015 IF T$<>"NULL " THEN GOTO 3050
3025 DC=IB:GOSUB 1000:GOSUB 13000
3030 PRINT AD$;"  ";HX$;"          *"
3040 AD=AD+1:GOTO 3000
3050 ON BY(IB) GOTO 3060,3090,4050
3060 DC=IB:GOSUB 1000:GOSUB 13000
3070 PRINT AD$;"  ";HX$;"          ";T$
3075 AD=AD+1
3080 GOTO 3000
3090 DC=IB:GOSUB 1000
4000 B1$=HX$
4010 DC=PEEK(AD+1):GOSUB 1000
4011 B2$=HX$
4024 GOSUB 13000:P=DC
4030 PRINT AD$;"  ";B1$;" ";B2$;"      ";T$;
     "  ";P
4035 AD=AD+2
4040 GOTO 3000
4050 DC=IB:GOSUB 1000
4060 B1$=HX$
4070 DC=PEEK(AD+1):GOSUB 1000
4080 B2$=HX$
4090 DC=PEEK(AD+2):GOSUB 1000
5000 B3$=HX$
5010 OP=PEEK(AD+1)+(PEEK(AD+2)*256)
5011 GOSUB 13000
5020 PRINT AD$;"  ";B1$;" ";B2$;" ";B3$;"  ";
     T$;"  ";OP
5025 AD=AD+3
5030 GOTO 3000
5050 INPUT T$
5051 IF T$<>"" THEN RETURN
5052 GRAPHICS 0:I=0:GOTO 3000
6000 DATA BRK  ,1,ORAIX,2,NULL ,0,NULL ,0,NULL ,0,
     ORAZ ,2,ASL  ,2,NULL ,0,PHP  ,1
6010 DATA ORAIM,2,ASLA ,1,NULL ,0,NULL ,0,ORA  ,3,
     ASL  ,3,NULL ,0,BPL  ,2,ORAIY,2
6020 DATA NULL ,0,NULL ,0,NULL ,0,ORAZX,2,ASLZX,2,
     NULL ,0,CLC  ,1,ORAY ,3
6030 DATA NULL ,0,NULL ,0,NULL ,0,ORAX ,3,ASLX ,3,
     NULL ,0,JSR  ,3,ANDIX,2,NULL ,0
6040 DATA NULL ,0,BITZ ,2,ANDZ ,2,ROLZ ,2,NULL ,0,
     PLP  ,1,ANDIM,2,ROLA  ,1,NULL ,0
6050 DATA BIT  ,3,AND  ,3,ROL  ,3,NULL ,0,BMI  ,2,
     ANDIY,2,NULL ,0,NULL ,0,NULL ,0
6060 DATA ANDZX,2,ROLZX,2,NULL ,0,SEC  ,1,ANDY ,3,
     NULL ,0,NULL ,0,NULL ,0,ANDX ,3
6070 DATA ROLX ,3,NULL ,0,RTI  ,1,EORIX,2,NULL ,0,
     NULL ,0,NULL ,0,EORZ ,2,LSRZ ,2
6080 DATA NULL ,0,PHA  ,1,EORIM,2,LSRA ,1,NULL ,0,
     JMP  ,3,EOR  ,3,LSR  ,3,NULL ,0
6090 DATA BVC  ,2,EORIY,2,NULL ,0,NULL ,0,NULL ,0,
     EORZX,2,LSRZX,2,NULL ,0
6100 DATA CLI  ,1,EORY ,3,NULL ,0,NULL ,0,NULL ,0,
     EORX ,3,LSRX ,3,NULL ,0,RTS  ,1
6110 DATA ADCIX,2,NULL ,0,NULL ,0,NULL ,0,ADCZ ,2,
     RORZ ,2,NULL ,0,PLA  ,1,ADCIM,2
6120 DATA RORA ,1,NULL ,0,JMPI ,3,ADC  ,3,ROR  ,3,
     NULL ,0,BVS  ,2,ADCIY,2,NULL ,0
6130 DATA NULL ,0,NULL ,0,ADCZX,2,RORZX,2,NULL ,0
     SEI  ,1,ADCY ,3,NULL ,0,NULL ,0
6140 DATA NULL ,0,ADCX ,3,RORX ,3,NULL ,0,NULL ,0,
     STAIX,2,NULL ,0,NULL ,0,STYZ ,2
6150 DATA STAZ ,2,STXZ ,2,NULL ,0,DEY  ,1,NULL ,0,
     TXA  ,1,NULL ,0,STY  ,3,STA  ,3
6160 DATA STX  ,3,NULL ,0,BCC  ,2,STAIY,2,NULL ,0,
     NULL ,0,STYZX,2,STAZX,2,STXZY,2
6170 DATA NULL ,0,TYA  ,1,STAY ,3,TXS  ,1,NULL ,0,
     NULL ,0,STAX ,3,NULL ,0,NULL ,0
6180 DATA LDYIM,2,LDAIX,2,LDXIM,2,NULL ,0,LDYZ ,2,
     LDAZ ,2,LDXZ ,2,NULL ,0
6190 DATA TAY  ,1,LDAIM,2,TAX  ,1,NULL ,0,LDY  ,3,
     LDA  ,3,LDX  ,3,NULL ,0,BCS  ,2
6200 DATA LDAIY,2,NULL ,0,NULL ,0,LDYZX,2,LDAZX,2,
     LDXZY,2,NULL ,0,CLV  ,1
6210 DATA LDAY ,3,TSX  ,1,NULL ,0,LDYX ,3,LDAX ,3,
     LDXY ,3,NULL ,0,CPYIM,2,CMPIX,2
6220 DATA NULL ,0,NULL ,0,CPYZ ,2,CMPZ ,2,DECZ ,2,
     NULL ,0,INY  ,1,CMPIM,2,DEX  ,1
6230 DATA NULL ,0,CPY  ,3,CMP  ,3,DEC  ,3,NULL ,0,
     BNE  ,2,CMPIY,2,NULL ,0,NULL ,0
6240 DATA NULL ,0,CMPZX,2,DECZX,2,NULL ,0,CLD  ,1,
     CMPY ,3,NULL ,0,NULL ,0,NULL ,0
6250 DATA CMPX ,3,DECX ,3,NULL ,0,CPXIM,2,SBCIX,2,
     NULL ,0,NULL ,0,CPXZ ,2,SBCZ ,2
6260 DATA INCZ ,2,NULL ,0,INX  ,1,SBCIM,2,NOP  ,1,
     NULL ,0,CPX  ,3,SBC  ,3,INC  ,3
6270 DATA NULL ,0,BEQ  ,2,SBCIY,2,NULL ,0,NULL ,0,
     NULL ,0,SBCZX,2,INCZX,2,NULL ,0,SED  ,1
6280 DATA SBCY ,3,NULL ,0,NULL ,0,NULL ,0,SBCX ,3,
     INCX ,3,NULL ,0
6290 DATA 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
9000 GRAPHICS 0
9010 PRINT "Enter Program Name";:INPUT N$
9015 N2$="C:":N2$(3)=N$
9020 OPEN #1,4,0,N2$
9030 INPUT #1,ZZ
9040 INPUT #1,EN
9050 FOR AD=ZZ TO EN
9060 INPUT #1,DA
9070 POKE AD,DA
9080 NEXT AD
9090 CLOSE #1
9100 RETURN
10000 GRAPHICS 0
10010 PRINT "Enter Address in Base 10";:INPUT AD
10015 IF AD>65535 THEN GOTO 10000
10020 T99=USR(AD)
10030 RETURN
11000 GRAPHICS 0
11010 PRINT "Enter Value to be passed";:INPUT AC
11020 PRINT "Enter Address in Base 10";:INPUT AD
11050 T99=USR(AD,AC)
11060 RETURN
13000 A=AD:S3=INT(AD/4096)
13002 A=A-S3*4096
13010 S2=INT(A/256)
13012 A=A-S2*256
13020 S=INT(A/16)
13060 U=AD-(S3*4096+S2*256+S*16)
13070 S3$=CO$(S3+1)
13080 S2$=CO$(S2+1)
13090 S$=CO$(S+1)
13100 U$=CO$(U+1)
13110 AD$(1)=S3$:AD$(2)=S2$:AD$(3)=S$:AD$(4)=U$
13120 RETURN
14000 GRAPHICS 0:AD=826:ZZ=826
14010 PRINT "(MNEMONIC)(SPACE)(OPERAND)"
14020 GOSUB 15000
14030 F=0
14040 FOR E=0 TO 255
14041 T$=MN$(E*5+1)
14050 IF T$<>M$ THEN GOTO 14060
14051 B=BY(E):F=1:CD=E:E=256
14060 NEXT E
14070 IF F=0 THEN GOTO 14260
14080 ON B GOSUB 14100,14130,14180
14090 GOTO 14020
```

```
14100 POKE AD,CD
14110 AD=AD+1
14120 RETURN
14130 IF OP>255 OR OP<0 THEN PRINT "ERROR -
      OPERAND":RETURN
14140 POKE AD,CD
14150 POKE AD+1,OP
14160 AD=AD+2
14170 RETURN
14180 IF OP>65535 OR OP<0 THEN PRINT "ERROR -
      OPERAND":RETURN
14190 POKE AD,CD
14200 B2=INT(OP/256)
14210 B1=OP-(B2*256)
14220 POKE AD+1,B1
14230 POKE AD+2,B2
14240 AD=AD+3
14250 RETURN
14260 IF (M$="ORG  ") OR (M$="END  ") OR
      (M$="DC   ") THEN GOTO 14280
14270 PRINT "ERROR - PSUEDO-OP":STOP
14280 IF M$="ORG  " THEN GOTO 14300
14290 GOTO 14340
14300 IF FO=1 THEN PRINT "ERROR - MULTIPLE
      ORG":GOTO 14020
14310 FO=1
14320 AD=OP:ZZ=OP
14330 GOTO 14020
14340 IF M$="END  " THEN GOTO 14360
14350 GOTO 14480
14360 EN=AD-1
14370 RETURN
```

```
14480 POKE AD,OP
14510 AD=AD+1
14520 GOTO 14020
15000 M$="     ":A$="":INPUT A$
15010 IF LEN(A$)<3 THEN PRINT "ERROR - LENGTH":
      GOTO 15000
15030 S=0:FOR M=1 TO LEN(A$)
15040 U$=A$(M):IF U$=" " THEN S=M:M=LEN(A$)
15050 NEXT M
15060 IF S=0 THEN GOTO 15100
15070 FOR M=1 TO S-1:U$=A$(M):M$(M)=U$:NEXT M
15072 IF S=6 THEN GOTO 15080
15074 FOR M=S TO 5:M$(M)=" ":NEXT M
15080 V$="":V$=A$(S):OP=VAL(V$)
15090 RETURN
15100 S=LEN(A$)+1
15110 FOR M=1 TO S-1:U$=A$(M):M$(M)=U$:NEXT M
15120 IF S=6 THEN GOTO 15090
15130 FOR M=S TO 5:M$(M)=" ":NEXT M:GOTO 15090
20000 GRAPHICS 0
20010 PRINT "Enter Program Name";:INPUT N$
20015 N2$="C:":N2$(3)=N$
20020 OPEN #1,8,0,N2$
20030 PRINT #1,ZZ
20040 PRINT #1,EN
20050 FOR AD=ZZ TO EN
20060 DA=PEEK(AD)
20070 PRINT #1,DA
20080 NEXT AD
20090 CLOSE #1
20100 RETURN
```

Function 2 allows you to save your program on tape. When you select this function you will be asked for the name you wish to save the program under. The console will beep twice and when you press RETURN your program will be saved on tape. (Disk owners will find it quite simple to alter this section of the assembler to save programs on disk.)

Function 3 is the opposite of function 2; it loads your program from tape. Once again you will have to enter a program name. If the program name you have entered is not found on the tape or disk an error will result.

Functions 4 and 5 allow you to execute your machine language object program. Function 4 simply jumps to the starting address you have entered. Function 5 allows you to pass a value to your program. Rather than confuse you about how this is done I'll refer you to the *Atari BASIC Reference Manual* for details.

Finally, function 6 allows you to list your program after it has been entered. This routine is actually a disassembler and you can use it to snoop around in Atari's ROMs just as easily. This function will display a screenful of disassembled code and then halt. To continue the listing press any key and RETURN. To halt, and go back to the main menu, simply press RETURN.

Some Changes to the Program

Since getting the listing presented here I have discovered that a couple of changes are in order.

Add line 12.

```
12 DIM N$(8),N2$(10)
```

This makes the entering of program names work correctly.

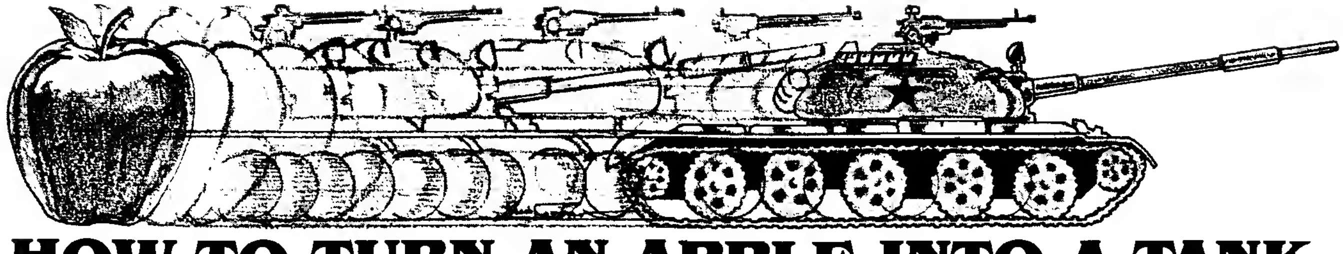
It is helpful to know where you can safely put information and programs into your machine's memory. In low memory it appears that these areas are free:

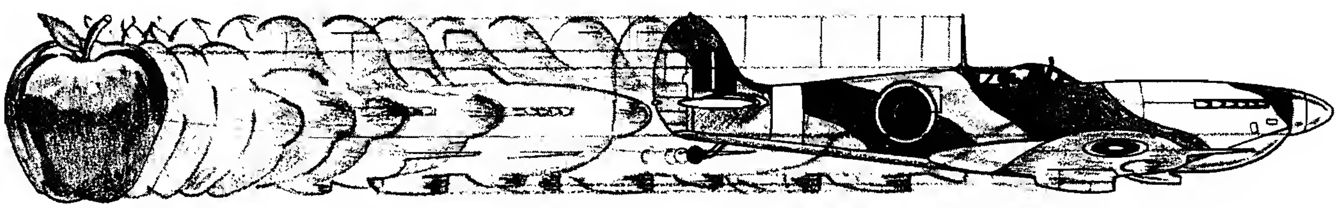
```
  21-64   (All addresses are in
           decimal)
 252-563
 713-740
1664-1791
```

Take reasonable care with where you put things and you should be OK.

William L. Colsher took an undergraduate major in Computer Science at Ohio State University. He worked for several years in mainstream data processing, leaving that field three years ago to write and consult in the microcomputer field. He is presently manager of the Oak Brook Computer Centre in Oakbrook Terrace, Illinois. He owns several small computer systems including a fully expanded TRS-80 Model I and a similarly configured Atari 800.

MICRO





Turning USR(X) Routines Into BASIC DATA Statements

This program saves machine language routines as BASIC DATA states. It also includes a hexadecimal to decimal converter.

Thomas Cheng
26 Madison Street Apt. 41
New York, New York 10038

If for any reason you are writing machine language subroutines for the Ohio Scientific C1P (or any other cassette Ohio Scientific machine), the first problem which will present itself is the lack of any method of saving the routine. An alternative to spending 5 to 10 minutes for the ASSEMBLER or the EXTENDED MONITOR to load in, then save using that, is to turn the routine into a series of DATA statements in BASIC. To reload these values, you would then load the DATA statements into program memory, then POKE them into the proper memory locations.

For example, if I add a machine language subroutine fifty bytes in length, the program would save: first, a line number followed by the DATA keyword, then the actual program in decimal format. The first such line saved will contain the two pointers for the location of the program in decimal format. I chose to output these numbers separately, so that I could easily change them.

The following is an illustration of the workings of the program.

```
FOR K=0 to 63:POKE
    K+4096,K: NEXTK
RUN
START, END? 4096, 4157
LINENO, INC? 10,10
```

Result

```
10 DATA 4096,4157
20 DATA 0,1,2,3,4,5,6,7,8,9,10,
        11,12,13,14,15
30 DATA 16,17,18,19,20,21,22,
        23,24,25,26,27,28,29,
        30,31
40 DATA 32,33,34,35,36,37,38,
        39,40,41,42,43,44,45,
        46,47
50 DATA 48,49,50,51,52,53,54,
        55,56,57,58,59,60,61,
        62,63
```

This is what the BASIC program should do, but due to the fact that I am extremely lazy, I have added in several frills to the program.

First and most important, is a hexadecimal to decimal converter inherent in the program. This conversion routine is located at lines 140 to 160, inclusive. This little routine has turned out to be quite handy, as I have used it

```
5 REM MACHINE LANGUAGE SAVE
7 REM ***THOMAS CHENG***
10 INPUT"START, END";B$,C$:INPUT"LINENO, INC";ST,IN
20 IFLEFT$(B$,1)="$"THENGOSUB140:B=A:GOTO40
30 B=VAL(B$):REM IT ALWAYS ENDS UP IN B
40 B$=C$:IFLEFT$(B$,1)="$"THENGOSUB140:GOTO55
50 A=VAL(C$):REM-A IS SECOND VALUE
55 SAVE:PRINT:PRINT:PRINTST;"READN,N2:FORK=NTON2:READQ:POKEK,Q:NEXTK"
60 ST=ST+IN:PRINTST;"Q=INT(N/256):POKE12,Q:POKE11,N-Q*256"
70 ST=ST+IN:PRINTST;"DATA";MID$(STR$(B),2);",";MID$(STR$(A),2)
80 ST=ST+IN:C=B+15:PRINTST;"DATA";:IFC>ATHENC=A
90 FORK=BTOC:LO=PEEK(K):GOSUB170:PRINTA$;:IFK<CTHENPRINT",";
110 NEXTK
120 IFC<>ATHENB=C+1:PRINT:GOTO80
130 PRINT:PRINT:PRINT:POKE517,0:END
140 A=0:A$="0123456789ABCDEF":FORK=1TOLEN(B$):FORL=1TO16
150 IFMID$(B$,K,1)=MID$(A$,L,1)THENA=A+(16↑(LEN(B$)-K)*(L-1))
160 NEXTL:NEXTK:RETURN
170 A$=MID$(STR$(LO),2):RETURN
```

in several programs I have written. The routine expects the value that is to be converted to be passed in B$. (This is the reason for so much confusion in lines 20-50. First, the value is placed in B$, then B$ is tested for a dollar sign. If the first character of the string is a dollar sign, the value is assumed to be in hexadecimal, then converted to decimal. The test is separate for each value, a worthwhile touch.)

The routine uses the mathematical definition of a base; that is, the sum of sixteen raised to the power of the digit's place (0,1,2...) multiplied by the value of the digit itself, (1 for 1, 2, for 2,...10 for A, 11 for B) for all the digits of the number.

The hexadecimal number by my method would be equal to (16**2)(10) + (16**1)(1) + (16**0)(11) = 2560 + 16 + 11 = 2587 in base 10.

Other little touches are contained in lines 55-60. Line 55 will output the requisite BASIC commands to POKE the statements into memory. Line 60 will set the values of the locations 11 and 12 as a pointer to the first byte of the subroutine, in standard 6502 format. That is, least significant byte first.

Line 70 produced the beginning and ending locations of the subroutine, while the actual core of the program is located at lines 80-130, with line 170 serving as a subroutine to strip off the extra space which is usually found in front of any number being printed out.

Two Last Notes

After the inquiry in line 10 for the starting line number, the user should turn on his recorder for the following DATA statements. Secondly, this program will turn out sixteen values per line of DATA statement. Occasionally, the numbers being printed out will run over the maximum line length of 72 characters, so the statement C = B + 15 in line 80 should be changed. The increment of 15 should be change to the true increment minus one.

Happy computing... and take a PEEK at some machine language routines!

MICRO

Improved Dual Tape Drive for SYM BASIC

These utility routines occupy less than one page of memory. However they greatly enhance the use of two cassettes, including the ability to automatically duplicate a tape full of BASIC programs.

George Wells
1620 Victoria Pl.
La Verne, California 91750

This article is an update to a previous article of mine which appeared in the November 1979 issue of MICRO (18:5). If you have that issue available you might want to review it, but it's not absolutely necessary since this article presents all new material and does not require any information contained in the original article. You may be wondering why the need for an "improved" article; why didn't I do it right the first time? Basically there are two reasons: Synertek System's new monitor 1.1 ROM and their Resident Assembler/Editor ROMs, both of which became available after I wrote the original article.

MON 1.1 indicates the ID of the tape file being loaded on the left-most digit of SYM's LED display; a very minor change from MON 1.0 but one that allows the program which called the tape load routine to determine the ID of the file just loaded. This is significant because it frees us from the drudgery of manually duplicating a tape full of BASIC programs by typing LOAD A, SAVE A, LOAD B, SAVE B, etc., as in the original article. By contrast, this article presents a program that will do this automatically, similarly to the way RAE-1 duplicates its tape files. Another advantage of this change is the ability of the calling program to provide true dual cassette control for BASIC programs. In the original article, the remote control for the write-only recorder will be turned on if the first file encountered during a load sequence is not the one asked for. Usually this won't matter since you don't normally have a tape ready to be written in the write-only recorder, but when you do, it's nice to have this added improvement.

RAE-1 provides for a second cassette control—but wouldn't you know—it requires a different port bit than the one I used originally (active low on PB7 of VIA #1 instead of active high on PB4 of VIA #3). If you have not yet implemented this second tape control, Synertek Systems Technical Note #101 describes one way to do it using relays. The program described here uses the same control as RAE-1, but since it does not require the RAE-1 ROMs, you can customize it to any control by modifying the TAPE.OFF.C and TAPE.ON.C routines.

The program presented in this article contains three entry points. The first one is used at the monitor level and approximates the .L2 command with zero, one or three parameters. It is called instead with .L3 and turns on the second recorder for loading hi-speed tape files. It can also be called as .U0, especially from the on-board hex keypad (USR 0). The second entry point is used while in BASIC and is called by a LOAD command. The third entry point is called from the monitor with .G 1F7B (or .G F7B for 4K) and is used to duplicate a tape of BASIC programs. Detailed instructions for each of these follows, but it is assumed that you have already loaded the OBJECT CODE into memory. If you don't have 8K of RAM then you will have to change the ten "1F" bytes to "0F" when you load it into a 4K system starting at $0F01. This can be done easily by first Depositing and Verifying the OBJECT CODE at $F01 and then doing a .M 1F,F01 – FFF followed by ten sets of 0FG (no <CR>'s). Verify checksum should then be 6DBA instead of 6E5A. The best arrangement is to put the code in your own PROM along with the appropriate automatic initialization code (not described here).

Using the .L3 Command

Step 1: Enter the following monitor command once after every reset:

.SD 1F11,A66D (or .SD F11,A66D for 4K.)

Step 2: Put a tape in the read-only recorder and press the play button. Use .L3 instead of .L2 and enter the parameters just as for .L2.

Note: If you enter only one parameter, the control for the write-only recorder will be energized after the first file has passed, if it is not the one specified in the parameter ID. This is unavoidable without completely rewriting the monitor tape load routine. Just don't turn on your write-only recorder when loading tape files from the monitor.

```
;************************************************************;
;                                                            ;
;       DUAL CASSETTE TAPE DRIVE FOR SYM-1 BASIC             ;
;                                                            ;
;       BY GEORGE WELLS          OCTOBER 4, 1980             ;
;                                                            ;
;************************************************************;


;       HARDWARE REQUIREMENTS:
;               SYM-1 WITH MON 1.1 ROM.
;               8K RAM (CAN BE RELOCATED FOR 4K; SEE TEXT).
;               BASIC V1.1 ROM.
;               WRITE-RECORDER WITH STANDARD CONTROL.
;               READ-RECORDER CONTROLLED BY LOW SIGNAL ON
;                PB7 OF VIA #1 (SAME AS RAE-1 REQUIRES).
;               TERMINAL.
;               RAE-1 V1.0 ROM OPTIONAL.


;       *** ZERO PAGE DEFINITIONS ***

SUP.PRINT  .DE $17      MS BIT SET SUPPRESSES PRINT
CRLF.NULLS .DE $18      NUMBER OF CR/LF NULLS
PRINT.POS  .DE $19      CURRENT COLUMN PRINT POSITION
WIDTH      .DE $1A      MAXIMUM WIDTH OF PRINT LINE
P1C        .DE $1C      COPY OF PARAMETER 1 (ID)
BUFAD      .DE $FE      MONITOR TAPE ROUTINE BUFFER ADDRESS

;       *** I/O PORT DEFINITIONS ***

OR1B       .DE $A000    DATA REGISTER FOR TAPE CONTROL
DDR1B      .DE $A002    DIRECTION REGISTER FOR TAPE CONTROL
DIG        .DE $A400    DATA REGISTER FOR DISPLAY DIGIT
DDRDIG     .DE $A401    DIRECTION REGISTER FOR DIGIT

;       *** SYSTEM RAM DEFINITIONS ***

TAPDEL     .DE $A630    TAPE DELAY LOCATION
P1         .DE $A64E    TAPE ID PARAMETER
P2         .DE $A64C    TAPE START ADDRESS
P3         .DE $A64A    TAPE STOP ADDRESS + 1
PARNR      .DE $A649    NUMBER OF PARMS IN MONITOR COMMAND

;       *** MONITOR 1.1 ROM DEFINITIONS ***

INCP3      .DE $8293    INCREMENT PARAMETER 3
CONFIG     .DE $89A5    CONFIGURE DISPLAY I/O
LOADT      .DE $8C78    LOAD TAPE ENTRY POINT
EX10       .DE $8D4E    STOP TAPE EXIT
START      .DE $8DA9    INITIALIZE AND START TAPE
DUMPT      .DE $8E87    DUMP TAPE ENTRY POINT
ACCESS     .DE $8B86    UNWRITE-PROTECT SYSTEM RAM

;       *** BASIC V1.1 ROM DEFINITIONS ***

1F4A- 48                     PHA
1F4B- 20 01 1F               JSR TAPE.OFF.C      TURN OFF READ-RECORDER
1F4E- 68                     PLA
1F4F- 60                     RTS

;    *** TO ACTIVATE BASIC LOAD USE 7937 MEMORY SIZE AND
;    *** ENTER:  POKE 202,80: POKE 203,31
```

```
1F50- AE 4E A6      BASIC.LOAD LDX P1                GET DESIRED ID
1F53- 86 1C                    STX *P1C              SAVE IT
1F55- F0 DF                    BEQ LOAD.TAPE         BRANCH IF ID = 0
1F57- E8                       INX
1F58- F0 DC                    BEQ LOAD.TAPE         BRANCH IF ID = $FF
1F5A- 20 BA 1F      BASIC.LOOP JSR LOAD.ANY          ELSE LOAD ANY FILE
1F5D- F0 0A                    BEQ BASIC.DONE        BRANCH IF "CR" ABORT
1F5F- 49 80                    EOR #$80              TEST FOR "END" TOKEN
1F61- F0 06                    BEQ BASIC.DONE        BRANCH IF SO
1F63- 49 80                    EOR #$80              RESTORE ID
1F65- 45 1C                    EOR *P1C              TEST FOR DESIRED ID
1F67- D0 F1                    BNE BASIC.LOOP        BRANCH IF NOT
1F69- 60            BASIC.DONE RTS                   CARRY SET MEANS BAD LOAD

1F6A- 00            ZERO.TABLE .BY 0         ;FLAG TO SUPPRESS BASIC PRINT
1F6B- 01                       .BY 1         ;NUMBER OF CR/LF NULLS
1F6C- 00                       .BY 0         ;COLUMN PRINT POSITION
1F6D- 48                       .BY 72        ;TERMINAL PRINT WIDTH

1F6E- A2 03         DUP.INIT   LDX #3                COPY FOUR BYTES FROM
1F70- BC 6A 1F      DUP.I.LOOP LDY ZERO.TABLE,X       TABLE TO
1F73- 94 17                    STY *SUP.PRINT,X       PAGE ZERO
1F75- CA                       DEX                    FOR BASIC PRINT CONTROL
1F76- 10 F8                    BPL DUP.I.LOOP
1F78- 4C 86 8B                 JMP ACCESS            ENABLE SYSTEM RAM & RETURN

                    ;    *** TO DUP A BASIC PROGRAM TAPE ENTER .G 1F7B
                    ;    *** SEE TEXT FOR ADDITIONAL INFORMATION

1F7B- 20 6E 1F      DUP.LEADER JSR DUP.INIT          INITIALIZE PRINT CONTROL
1F7E- A2 0A                    LDX #10               FIRST FILE HAS LONG LEADER

1F80- 8E 30 A6      DUP.LOOP   STX TAPDEL            UPDATE TAPE DELAY
1F83- 20 BA 1F      DUP.LOAD   JSR LOAD.ANY          GET NEXT FILE FROM MASTER
1F86- 90 18                    BCC DUP.GOOD.L        BRANCH IF GOOD LOAD

1F88- 00                       BRK                   ELSE BREAK TO MONITOR
1F89- 20 6E 1F      DUP.MIDDLE JSR DUP.INIT          ENTRY TO RETURN FROM BREAK
1F8C- AA                       TAX                   TEST FOR ID = 0
1F8D- F0 F4                    BEQ DUP.LOAD          BRANCH IF SO
1F8F- 85 1C                    STA *P1C              ELSE SEARCH FOR ID = ACC
1F91- 20 BA 1F      DUP.REGET  JSR LOAD.ANY          GET NEXT FILE FROM MASTER
1F94- D0 04                    BNE DUP.NOABRT        BRANCH IF NOT "CR" ABORT
1F96- 00                       BRK                   ELSE BREAK AGAIN TO MONITOR
1F97- 20 6E 1F                 JSR DUP.INIT          INITIALIZE PRINTER CONTROL
1F9A- B0 F5         DUP.NOABRT BCS DUP.REGET         BRANCH IF BAD LOAD
1F9C- C5 1C                    CMP *P1C              TEST FOR DESIRED ID
1F9E- D0 F1                    BNE DUP.REGET         BRANCH IF NOT

1FA0- 8D 4E A6      DUP.GOOD.L STA P1                PASS ID TO SAVE ROUTINE

                    KEY.WORDS  .DE $C089     TABLE OF BASIC KEY WORDS
                    PRINT.SP   .DE $C971     BASIC PRINT SPACE
                    PRINT.?    .DE $C974     BASIC PRINT QUESTION MARK
                    BASIC.PRNT .DE $C976     BASIC PRINT CHARACTER IN ACC

                    ;       *** RESIDENT ASSEMBLER/EDITOR ROM DEFINITIONS ***
                    ;       NOTE:
                    ;               THE FOLLOWING TWO ROUTINES CAN BE USED
                    ;                INSTEAD OF TAPE.OFF.C AND TAPE.ON.C
                    ;                IF RAE-1 V1.0 IS ALSO AVAILABLE.
```

(continued)

```
                        TAPE.OFF    .DE $E318    TURN OFF SECOND RECORDER
                        TAPE.ON     .DE $E32A    TURN ON SECOND RECORDER

                                    .BA $1F01
                                    .OS

1F01- AD 00 A0          TAPE.OFF.C LDA OR1B           TURN OFF READ-ONLY RECORDER
1F04- 09 80                        ORA #%10000000
1F06- D0 05                        BNE TAPE.CNTRL     (ALWAYS)

1F08- AD 00 A0          TAPE.ON.C  LDA OR1B           TURN ON READ-ONLY RECORDER
1F0B- 29 7F                        AND #%01111111
1F0D- 8D 00 A0          TAPE.CNTRL STA OR1B
1F10- 60                           RTS

                        ;   *** .L3 COMMAND SIMULATES .L2 WITH SECOND RECORDER.
                        ;   *** TO ACTIVATE ENTER:   .SD 1F11,A66D

1F11- C9 14             L3.COMMAND CMP #$14           TEST FOR L3 HASH CODE
1F13- D0 14                        BNE L3.ERROR       BRANCH IF NOT

1F15- AE 49 A6                     LDX PARNR          GET NUMBER OF PARAMETERS
1F18- F0 1A                        BEQ LOAD.HI.SP     BRANCH IF NO PARAMETERS
1F1A- E0 02                        CPX #2             TEST FOR 2 PARAMETERS
1F1C- F0 0B                        BEQ L3.ERROR       BRANCH IF SO
1F1E- B0 0B                        BCS LOAD.3PARM     ELSE BRANCH IF 3 PARAMETERS
1F20- AE 4A A6                     LDX P3             ELSE PUT SINGLE PARAMETER
1F23- 8E 4E A6                     STX P1              WHERE IT BELONGS
1F26- E8                           INX                TEST FOR PARAMETER ID = $FF
1F27- D0 0B                        BNE LOAD.HI.SP     BRANCH IF NOT

1F29- 38                L3.ERROR   SEC                ELSE SET ERROR FLAG AND
1F2A- 60                           RTS                 RETURN TO MONITOR

1F2B- AE 4E A6          LOAD.3PARM LDX P1             TEST FOR PARAMETER ID = $FF
1F2E- E8                           INX
1F2F- D0 F8                        BNE L3.ERROR       BRANCH IF NOT
1F31- 20 93 82                     JSR INCP3          ELSE INCREMENT PARAMETER 3

1F34- A0 80             LOAD.HI.SP LDY #$80           HI SPEED TAPE LOAD ENTRY
1F36- 20 A9 8D          LOAD.TAPE  JSR START          INITIALIZE TAPE ROUTINE
1F39- 20 4E 8D                     JSR EX10           TURN OFF WRITE-RECORDER
1F3C- AD 02 A0                     LDA DDR1B          SET READ-RECORDER CONTROL
1F3F- 09 80                        ORA #%10000000     BIT TO OUTPUT
1F41- 8D 02 A0                     STA DDR1B
1F44- 20 08 1F                     JSR TAPE.ON.C      TURN ON READ-RECORDER
1F47- 20 7B 8C                     JSR LOADT+3        CONTINUE TO LOAD TAPE
1FA3- 48                           PHA                SAVE ID FOR LATER TEST
1FA4- A9 02                        LDA #2             ALL PROGRAMS START AT $0201
1FA6- 8D 4D A6                     STA P2+1           TAPE START ADDRESS HI = 2
1FA9- 4A                           LSR A              CHANGE 2 TO 1
1FAA- 8D 4C A6                     STA P2             TAPE START ADDRESS LO = 1
1FAD- A0 80                        LDY #$80           SET HI SPEED TAPE MODE
1FAF- 20 87 8E                     JSR DUMPT          SAVE FILE ON WRITE-RECORDER
1FB2- A2 04                        LDX #4             SET TAPE DELAY TO DEFAULT
1FB4- 68                           PLA                GET TAPE FILE ID
1FB5- 49 80                        EOR #$80           TEST FOR "END" TOKEN
1FB7- D0 C7                        BNE DUP.LOOP       BRANCH IF NOT
1FB9- 60                           RTS                RETURN TO MONITOR (CARRY=0)
```

```
1FBA- A9 00        LOAD.ANY    LDA #0              SET UP TO GET ANY FILE
1FBC- 8D 4E A6                 STA P1               FROM TAPE
1FBF- 20 34 1F                 JSR LOAD.HI.SP      HI SPEED MODE ONLY
1FC2- 90 04                    BCC LOAD.GOOD       BRANCH ON GOOD LOAD
1FC4- 49 8C                    EOR #$8C            IF ABORT, RETURN ID = $00
1FC6- F0 37                    BEQ LOAD.DONE        AND TAKE BRANCH
1FC8- 08           LOAD.GOOD   PHP                 SAVE CARRY
1FC9- 4E 01 A4                 LSR DDRDIG          CHANGE FROM $FF TO $7F
1FCC- AD 00 A4                 LDA DIG             GET 7 LS BITS OF ID
1FCF- 0A                       ASL A
1FD0- EE 01 A4                 INC DDRDIG          CHANGE FROM $7F TO $80
1FD3- 0E 00 A4                 ASL DIG             GET MS BIT OF ID
1FD6- 6A                       ROR A               COMBINE ALL 8 BITS OF ID
1FD7- 48                       PHA                 SAVE ID FOR LATER
1FD8- 49 80                    EOR #%10000000      CHANGE MSB FOR LATER BRANCH
1FDA- 30 10                    BMI LOAD.PRINT      BRANCH IF NOT TOKEN
1FDC- AA                       TAX                 X COUNTS THRU KEYWORDS
1FDD- A0 FF                    LDY #$FF            Y COUNTS THRU CHARACTERS
1FDF- C8           LOAD.TOKEN  INY                 STEP TO NEXT CHARACTER
1FE0- B9 88 C0                 LDA KEY.WORDS-1,Y   LAST CHARACTER IN EACH
1FE3- 10 FA                    BPL LOAD.TOKEN       KEYWORD IS MINUS
1FE5- CA                       DEX                 X GOES MINUS JUST BEFORE
1FE6- 10 F7                    BPL LOAD.TOKEN       PROPER KEYWORD IS REACHED
1FE8- C8           LOAD.NCHAR  INY                 STEP TO NEXT CHARACTER
1FE9- B9 88 C0                 LDA KEY.WORDS-1,Y
1FEC- 20 76 C9     LOAD.PRINT  JSR BASIC.PRNT      PRINT CHARACTERS UNTIL
1FEF- 10 F7                    BPL LOAD.NCHAR       LAST MINUS CHAR IS REACHED
1FF1- 68                       PLA                 PUT ID CHARACTER OR
1FF2- A8                       TAY                  TOKEN IN Y
1FF3- 28                       PLP                 GET CARRY
1FF4- 08                       PHP
1FF5- 90 03                    BCC LOAD.SPACE      BRANCH IF GOOD LOAD
1FF7- 20 74 C9                 JSR PRINT.?         ELSE, PRINT QUESTION MARK
1FFA- 20 71 C9     LOAD.SPACE  JSR PRINT.SP        PRINT SPACE
1FFD- 28                       PLP                 RESTORE CARRY
1FFE- 98                       TYA                 RESTORE Z FLAG
1FFF- 60           LOAD.DONE   RTS                 RETURN WITH ID IN A

                               .EN
```

Using the BASIC LOAD Command

Step 1: Jump to BASIC and use 7937 in response to MEMORY SIZE? (3841 for 4K).

Step 2: Enter the following direct command:

POKE 202,80: POKE 203,31

or if you have 4K:

POKE 202,80: POKE 203,15.

Now you can use the LOAD command just as before. However, as each file is read its ID is echoed to your terminal followed by a space. If the file has a load error, a question mark is printed immediately after the ID and the search continues until the correct ID is found.

You can abort the tape load process by hitting "CR" on the hex keypad while the search sync character is being displayed. The tape load will also automatically abort if an "END" file is read. Such a file is created by entering NEW and then SAVE END. Normally, the tape ID is the ASCII equivalent of the character entered immediately after a SAVE. However, the BASIC tokens are also allowed as valid tape IDs by entering one of the reserved words listed on page 9 of the BASIC manual. (GET and GO should be added to the list.) Thus you can have a program called GO, LIST, DATA, or even SAVE. For example, type SAVE LIST and LOAD LIST. The key word END is reserved for the last program on the tape.

Using the Program Duplicator

Step 1: Rewind your BASIC program master tape and ready it in your "play" or read-only recorder.

Step 2: Rewind a blank cassette and ready it in your "record" or write-only recorder.

Step 3: From the monitor, enter the following command.

.G 1F7B (or .G F7B for 4K).

This will duplicate all of the BASIC programs from the master tape onto the blank tape. It will even use a long tape delay before the first program to get the tape off its leader and it will quit after the "END" file has been copied.

Step 4: In case an error is detected while loading a program, the program will break to the monitor and display 1F89,0 (or 0F89,0 for 4K). At this point you should put the read-only recorder in rewind and enter a .G command. After the tape has gotton past the file which caused the error, put the read-only recorder back in play mode. You don't need to worry about rewinding too far since the program will only accept the same file ID as the one which originally caused the error. Subsequent tape errors will not cause the program to break again until the specified file has been correctly loaded and duplicated. However, you can cause a break by hitting "CR" on the hex keypad during sync search. After several attempts to load a defective file, remove your master tape cassette and substitute a backup in the read-only recorder until the file is loaded correctly and the write-only recorder turns on. Then re-install the master tape and allow the duplication process to continue.

Step 5: Any time the sync search indication is on, you can cause a break to the monitor by hitting "CR" on the hex keypad. (You may have to turn the read-only recorder off or on to make this work.) A .G command will allow you to continue, but you can also force the program to search for a specific file ID and continue duplicating by entering a .R command followed by three spaces, and changing the value of the accumulator register to the hex equivalent of the desired ID. A value of zero will allow any ID. Finally go back to the program with:

```
.G 1F89 (or .G F89 for 4K).
```

Step 6: After duplicating the entire tape, you should verify that the new copy can be loaded correctly. It is safest to keep two backup copies of your master tape which are known to be loadable, so that if for some reason while you are duplicating with any two of your three tapes and a file is impossible to load, you will still have a backup available. If you had only one master and one backup and a file became unloadable while duplicating, there is a good chance that the same file on the backup tape will be overwritten by a previous file during the duplication process, especially if you have updated an earlier portion of the master tape. The procedure I have followed, without losing any programs, is to copy the master tape onto backup number 1, then backup number 1 onto backup number 2, then backup number 2 onto the master, and finally verify the master by jumping to BASIC and doing a LOAD END, checking that all the IDs are printed without a question mark after them.

Actually, this is an over-simplification. In reality, the original master will consist of more than one tape from which the programs will eventually be combined onto the new master. This duplicating program was specifically designed to allow updating of a master tape primarily through the technique described in *step 5*. Of course, you can still take advantage of the dual cassette control and update a master tape manually through the BASIC interpreter using a sequence of LOAD A, modify, SAVE A, etc.

Explanation of Program Operation

The ASSEMBLY LISTING contains comments which should help in understanding how the program works. However, the following additional comments may also be helpful.

You may want to modify the program slightly for your particular needs. Locations $1F7F and $1FB3 contain tape delay values which control the number of sync bytes at the beginning of each tape file and therefore the length of time before each file. The first of these is used only for the first file on the tape and is a large number to allow time for the tape to get past its leader before recording the actual file. With the default High Speed Tape Waveform values this time is 15 seconds. However, if you use speeded-up Waveform values, you will want to use larger tape delay values to get the same time delays. I first learned about the ability to double or triple the cassette baud rate from *SYM-PHYSIS* 3-3 and I now use HSBDRY = $1A, TAPET1 = $20 and TAPET2 = $10 which allows an 8K file to be loaded or saved in about 20 seconds, instead of one minute. I also use $1E and $08 for the two tape delay constants in my version of the dual cassette program which I have put into EPROM.

If your SYM-1 system also has the RAE-1 ROMs available, you can use the TAPE.OFF and TAPE.ON routines in them and save 16 bytes of program space. This requires changing the subroutine calls at $1F45 and $1F4C. I have piggy-backed the two BASIC ROMs into socket U21 and the two RAE-1 ROMs into socket U22 in order to fit them all onto the SYM-1 at the same time, and still have one socket left for an EPROM. This requires bending pin 20 of the top ROMs and wiring them directly to their chip select decodes, along with two extra 3.3K pullup resistors.

Two other bytes which you may want to modify are located at $1F6B and $1F6D. The first one controls the number of nulls that are inserted by the BASIC print routines after each CR/LF and the second one controls the width of the print line. I have used values that are suitable for a teletype, but if your terminal has fewer characters per line you should put the hex equivalent in the second location. These values are used only by the BASIC Program Duplicator routine.

Don't try to use the Duplicator to copy machine language programs or files other than BASIC programs. The Duplicator assumes that the tape file starts at $201 which is true for all BASIC programs, and if you try to dup a file that started somewhere else, the program would still use $201 as the start of the file.

You should also be careful not to read a tape that contains files other than BASIC programs with the LOAD command, even if the file IDs are different from the one specified in the LOAD command. Under the original LOAD, if the ID did not match, the file would not be loaded. But with this new LOAD, every file that is encountered on the tape is loaded before the ID comparison is made.

If you have a special machine language program that vectors through the BASIC zero page jump instruction at $C9 (decimal 201) and specifies a tape ID of either $00 or $FF, then the second cassette control will be activated and the load routine will behave exactly as before.

Once you get a taste for dual cassette control, it's hard to live with only one. It's really worth installing in your SYM!

George Wells has been working on several utility-type software and hardware projects for his SYM such as the one described in this article. His latest project is a hardware design to interface a light pen to Texas Instruments' new TMS9918A high-resolution color video display processor chip. One of these days he thinks he might actually have time to write some programs to put these utilities to good use!

MICRO

In The Heart Of Applesoft

This article is not written to explain how Applesoft works, but to explain how to work with Applesoft, or more specifically, how and when to use (numerical) Applesoft routines. As an example, a matrix multiplication program is presented. This program runs on the average 5 times faster (depending on the numbers in the matrices) than a comparable BASIC program.

C. Bongers
Erasmus University
Postbus 1738
3000 DR Rotterdam
The Netherlands

My primary motivation to buy a microcomputer was to develop a number of statistical programs which were to be used for a research project I was working on. After comparing several micros with each other, with respect to execution speed of BASIC programs and expansion possibilities, my choice fell on the Apple II. The Apple which I bought was delivered with Applesoft BASIC in ROM. After studying the manuals, I started writing some test programs in order to learn the possibilities and the peculiarities of the machine. Among others, I wrote a program to generate all the permutations of a given sequence of symbols (for instance, ABC has the permutations ABC, ACB, BAC, BCA, CAB, CBA) and a program to solve the 10 by 6 pentomino puzzle (see *BYTE*, Nov. 1979). The permutation program ran reasonably fast (20 permutations per second) but the pentomino program turned out to be a disappointment. After waiting for several hours, it finally produced the first of the 2,339 solutions, so I never bothered trying to find more solutions.

Example 1

	Exponent	Mantissa					Sign
MFP:	$9D	$9E	$9F	$A0	$A1	$AC	$A2
Hex contents	81	F3	B0	00	00	C0	70

Decimal $2^{129-128}\left[15\times\frac{1}{16}+3\times\frac{1}{256}+11\times\frac{1}{4096}+0\times\ldots+12\times\frac{1}{16^{9}}\right]=1.90380859$

***Table 1:* Applesoft Routines**

I. General remarks

A. Notation

1. (→ A,Y) means: pointed to by Accumulator (low) and Y register (high).

2. (A = Z.F. = $X) means: Accumulator has to contain, or contains the contents of location $X. If $X equals zero, the zero flag (Z.F.) is set or must be set, otherwise the zero flag is or must be clear.

B. Remarks

1. For some routines presented below, the entry and/or exit values of the Accumulator, the X register and the Y register are given. If no entry is specified, no entry is necessary. If no exit or not all registers of an exit are specified, the registers not specified may have unpredictable values after the execution of the Applesoft routine.

2. For each routine, the memory locations that may be modified by the (error free) execution of the routine are given.

C. Warning

1. When working with m.l. programs that are called from BASIC, one may wish to use zero page locations to store temporary results. However, a number of zero page locations are initialized to certain values at the cold/warm start of Applesoft and changing the contents of these locations may lead to unexpected results. Furthermore, there are a number of locations in which Applesoft stores information during the execution of the BASIC program, such as the current line number or the pointer to the line from which data is being read. Clobbering one of these locations usually has the effect that the program will crash sooner or later.

In order to avoid problems when working with zero page addresses it is therefore recommended to consult the zero page usage map in the Applesoft manual first. (See also the memory atlas contructed by Prof. W.F. Luebbert, published in the August 1979 issue of MICRO.)

(continued)

At that time, however, I discovered that the Apple can also be programmed rather easily in machine language with the help of the mini-assembler. Since I was interested to know what speed gain could be obtained, I translated the permutation program in machine code. To my surprise, the program ran about 675 times faster (approximately 13,500 permutations per second) than the BASIC permutation program. Of course, I immediately got my pentomino program and translated this in machine code too. The 2,339 solutions now came out in less than 3 hours, which meant also a considerable gain in speed as compared to the BASIC program.

When to Use Machine Language Programs or Subroutines

Some programs, like those mentioned above, can easily be translated from BASIC to machine code. However, for the majority of the programs that I intend to write, this is not the case, since in these programs floating point variables rather than "one byte" variables have to be used. For some floating point arithmetic such as addition and multiplication, it is probably possible to write the routines yourself, but for functions such as the sine and the logarithm this would mean a lot of work. Furthermore, being busy with "trying to reinvent the wheel"is not a very stimulating idea.

However, there is a fairly easy way out of this problem. All the routines needed for floating point arithmetic, have to be somewhere in the Applesoft ROM, so all one has to do is list Applesoft and try to understand how it works. After locating the entries of the floating point routines, these routines then can be called by the machine language (m.l.) program. Although the whole process can be written down in a few lines, it took me several weeks of hard work before I knew enough of Applesoft to write, as an exercise, a matrix multiplication subroutine in m.l., which can be called from BASIC by means of the & symbol. This program runs about 8 times faster than a BASIC matrix multiplication subroutine and further has the advantage that the names of the matrices can be passed in an easy way. On the other hand, a disadvantage is that the m.l. program uses more memory space than the BASIC program. At the end of this article, the matrix multiplication program will be more extensively discussed.

Table 1: Applesoft Routines (continued)

Of course, this warning does not apply to (most of the) zero page locations that may be modified by the routines described below. For instance, if one uses neither the power function nor SQR nor trigonometrical functions, it will be safe to use locations $8A-$8E, since these locations are used by none of the other functions (routines) listed in this table.

2. If neither strings nor high-resolution graphics nor ON ERR statements are used, one can (probably) safely store temporary results in the following zero page locations:

   $6-$9, $17-$1F, $58-$5D, $71-$72, $CE-$D5, $D7, $D9-$EF, $F4-$FF

II. *Description and entries of the routines*

A. Charget-Charcheck

1. Purpose

   The memory locations $B8 and $B9 contain—during the execution of a BASIC program—a text pointer which points to the last retrieved character of the BASIC program. The Charget routines can be used to load the next character or the current character (again) in the Accumulator. To determine whether the character equals a predetermined symbol one of the Charcheck routines may be used.

2. Charget routines

   $B1: Advance text pointer and load next character in the Accumulator (spaces are ignored).

   Exit(A = next character,X = entry,Y = entry).

   Exit Status: Carry is clear if character is a digit (hex value:30-39), otherwise carry is set. Zero flag is set if character equals 0 (= end of line sign) or 3A (= end of statement sign, i.e. ":"), otherwise zero flag is cleared.

   Modifies $B8,$B9.

   $B7: Load current character another time in the Accumulator.

   Exit(A = current character,X = entry,Y = entry).

   For status see subroutine $B1.

3. Charcheck routines

   $E07D: Check whether character in Accumulator is a letter.

   Entry(A),Exit(A = entry,X = entry,Y = entry).

   Exit Status: Carry is set if character is a letter, otherwise carry is cleared.

   The following 4 routines can be used to check whether the text pointer points to a specific symbol. If the result of the check is positive, the next character is loaded in the Accumulator by means of the execution of subroutine $B1. In the other case, the message "SYNTAX ERROR" is displayed and Applesoft returns to BASIC command level. The exits of the 4 routines are:

   Exits(A = next character,X = entry,Y = 0), modify $B8,$B9.

   $DEC0: Check whether the character that is pointed to by the text pointer equals the character in the Accumulator.

   Entry (A).

   $DEB8: Check whether the text pointer points to a right parenthesis.

   $DEBB: Check whether the text pointer points to a left parenthesis.

   $DEBE: Check whether the text pointer points to a comma.

B. Compare

1. Purpose

The compare routines can be used for comparing a real variable in the MFP with a real variable in the SFP or a real variable in memory.

2. Compare routines

$DF6A: Compare MFP with SFP according to the status of the comparison in location $16. The result of the comparison (1 if true, 0 if false) is converted to a real variable in the MFP. The various types of comparisons are listed below.

Type of Comparison	$16 has to be put equal to:	Result comparison
>	1	1 if SFP > MFP, else 0
=	2	1 if SFP = MFP, else 0
<	4	1 if SFP < MFP, else 0
> =	3	1 if SFP ≥ MFP, else 0
< >	5	1 if SFP ≠ MFP, else 0
< =	6	1 if SFP ≤ MFP, else 0

Modifies $60,$61,MFP,SFP.

$EBB2: Compare MFP with memory (→ A,Y).

Entry(A,Y), Exit(A = FF if MFP < memory, A = 0 if MFP = memory, A = 1 if MFP > memory), modifies $60,$61.

C. Conversion

1. Purpose

The Conversion routines can be used to convert:

a) a real in the MFP to an integer
b) a one or two byte integer to a real in the MFP

Unless specified otherwise, all integers are assumed to be two's complement integers.

2. Real to integer conversion routines

$EBF2: Convert MFP to integer. The number in the MFP must be between -2^{31} and 2^{31} (notation: $-2^{31} <$ MFP $< 2^{31}$). Result is stored in mantissa of MFP (locations $9E-$A1).

Exit(Y = 0), modifies MFP.

$E752: Convert MFP, where $-2^{16} <$ MFP $< 2^{16}$, to two byte integer. Store result in $50 (low) and $51 (high).

Exit(A = $51,Y = $50), modifies $50,$51,MFP.

Remark: "Wrap around" occurs if the absolute value of the number in the MFP is larger than $2^{15} - 1$.

$E10C: Convert MFP, where $-2^{15} <$ MFP $< 2^{15}$, to two byte integer. Store result in $A0 (high) and $A1 (low).

Exit (Y = 0), modifies $60,$61,MFP.

$E108: Same as $E10C, except that entry-value of MFP must be: $0 \leq$ MFP $< 2^{15}$.

$DA65: Pack extension byte in MFP and convert MFP, where -2^{15} < MFP $< 2^{15}$, to two byte integer. Store integer (high byte first) in (→$85,$86).

Exit(Y = 1), modifies $60,$61,MFP.

(continued)

An important point to note is that, as a consequence of using floating point arithmetic, there is a significant drop of the speed gain, namely from a factor 675 obtained with the permutation program to a factor 8 obtained with the matrix multiplication program. The reason is that—when multiplying matrices—a relatively large portion of the CPU time is used for the multiplication and addition of floating point numbers. Whether this is done under control of a BASIC program, or by calling the appropriate routines in Applesoft from a m.l. program, makes no difference, since in both cases the same multiplication and addition routines are used. The gain of speed that occurs in the m.l. matrix multiplication program is obtained by short-cutting the time-consuming determination of the pointers to array elements in BASIC.

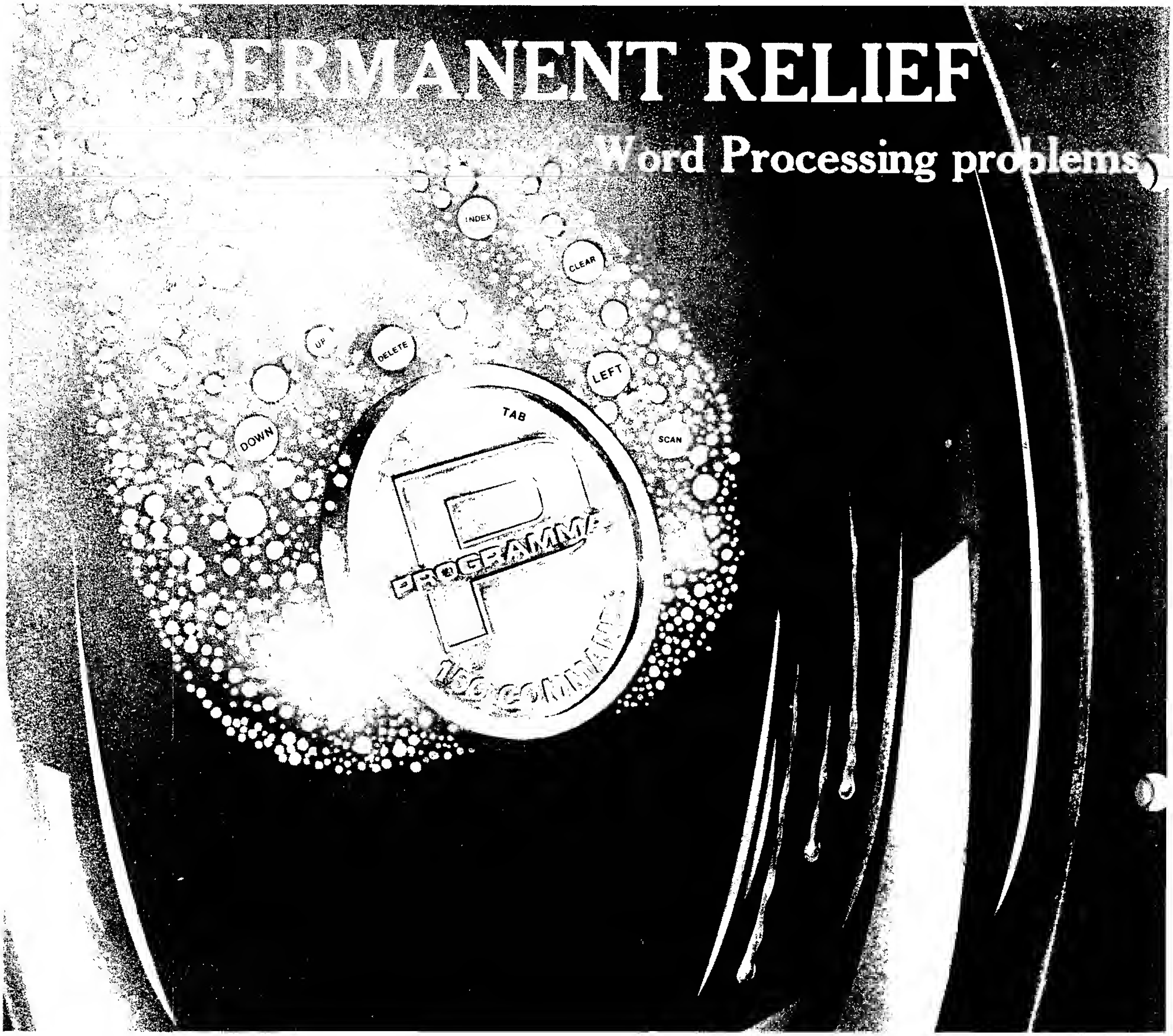
It will now also be clear that it does not make any sense to calculate for instance, 1000 logarithms by means of a m.l. program. When written in BASIC, thus

```
10 FOR I = 1 TO 1000 :
   A = LOG (I) : NEXT
```

the program will run approximately 23 seconds. About 90% of this time, the computer will be busy with the calculation of the logarithms, and about 10% of the time with the parsing of the statements and the evaluation of the FOR...NEXT loop. When writing a m.l. program to calculate the logarithms, one may expect it to run no more than 10% faster than the BASIC program, since as to the calculation of the logarithms, no time can be saved.

Therefore, with respect to gaining speed, it is only profitable to write a m.l. program or subroutine if, in this way, time-consuming access to array elements can be short-cutted or iterative parts of the program can be made more efficient. Some examples where m.l. routines will be useful are: finding the largest element of an array, calculating the inverse of a matrix, sorting the elements of a vector, or calculating probabilities under a bivariate (log) normal distribution.

Apart from gaining speed, there may however be other arguments for writing m.l. routines. For instance, one may wish to extend tape or disk versions of Applesoft with some self-written BASIC commands or functions. Also, it can be attractive to make frequently used subroutines more independent of



Apple PIE + Formatter

Apple PIE (Programma International Editor) and FORMAT (text formatter) offer full strength solutions to today's word processing problems. These versatile, powerful programs provide document preparation and word processing capabilities previously found only on much larger computer systems.

PIE is a general purpose, full screen editor that uses control keys and function buttons to provide a full range of editing capabilities such as search and replace, delete, copy, insert, move. Changes may be made directly anywhere on the screen and are shown as they are performed.

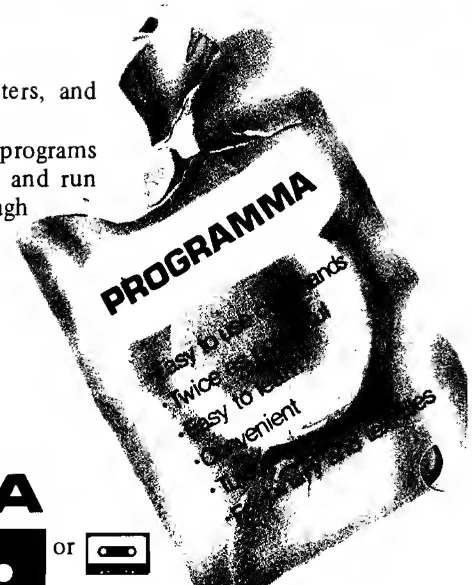
FORMAT uses simple instructions embedded in the input text to describe the desired appearance of the final document. It handles centering, underlining, indenting, page numbering, margins, headers, footers, even form letters, and includes a proofing capability.

These high-quality, cost-effective programs come with comprehensive documentation and run on a 32K Apple II. They are available through your local computer store or direct from Programma International, Inc. at the introductory price of $79.95*.

VIDEX VERSION T.M.
DOUBLE VISION T.M.
SUPR TERM VERSION T.M.
STANDARD VERSION

*December 1, $129.95.

11



PROGRAMMA

3400 Wilshire Boulevard
Los Angeles, California 90010
(213) 384-0579

or

Simple enough for the beginner. Versatile enough for the professional.

Table 1: Applesoft Routines (continued)

3. Integer to real conversion routines

$E2F2: Convert two byte integer in A (high) and Y (low) to real in MFP.

Entry(A,Y), Exit(Y=0), modifies MFP, puts $11 equal to zero.

$E301: Convert one byte integer in Y to positive real in MFP. (The integer in Y is thus not interpreted as a two's complement integer.)

Entry(Y), Exit(Y=0), modifies MFP, puts $11 equal to zero.

$EB93: Convert one byte integer in Accumulator to real in MFP.

Entry(A), Exit(Y=0), modifies MFP.

$DEE9: Pull integer (%) variable from memory (→$A0,$A1) into A (high) and Y (low). Next, convert integer to real in MFP.

Exit(Y=0), modifies MFP, puts $11 equal to zero.

D. Copy

1. Purpose

The Copy routines can be used to

a) pull data (from memory) into the MFP or the SFP
b) pack the MFP and store the MFP in memory
c) copy the MFP into the SFP and vice versa
d) push the MFP on stack or pull the SFP from stack

The Copy routines are for real variables only. For routines that handle integer (%) variables see Conversion.

2. MFP routines

$EAF9: pull memory (→ A,Y) into the MFP and put the extension byte equal to zero.

Entry(A,Y), Exit(A=Z.F.=$9D,X=entry,Y=0), modifies $5E,$5F,MFP.

$EAFD: Pull memory (→ $5E,$5F) into the MFP and put the extension byte equal to zero.

Exit(A=Z.F.=$9D,X=entry,Y=0), modifies MFP.

$DE10: Pack extension byte in MFP and push MFP on stack (6 bytes).

Exit(A=Z.F.=$9D), modifies $5E,$5F,MFP.

The following four routines pack the sign and the extension byte in the MFP, store the MFP in the locations indicated and put the extension byte equal to zero.

For all four routines the exits are:

Exits(A=Z.F.=$9D,Y=0), modify $5E,$5F, MFP.

$EB1E: store MFP in $98-$9C

$EB21: store MFP in $93-$97

$EB27: store MFP in (→ $85,$86)

$EB2B: store MFP in (→ X,Y)

3. SFP routines

$E9E3: Pull memory (→ A,Y) in the SFP and determine $AB (= the exclusive OR of the signs of the numbers in the MFP and the SFP).

Entry(A,Y), Exit (A=Z.F.=$9D,X=entry,Y=0), modifies $5E,5F,SFP,$AB.

$E9E7: Pull memory (→ $5E,$5F) in the SFP and determine $AB.

Exit(A=Z.F.=$9D,X=entry,Y=0), modifies SFP,$AB.

(continued)

the main program, so that parameters can be passed by value rather than by name, which in BASIC is only possible by means of a lot of PEEKs and POKEs. Last but not least, one may like the challenge involved in writing m.l. programs.

The Main and Secondary Floating Point Accumulator

Before presenting the Applesoft routines that can be of help when writing m.l. programs, the main and secondary floating point accumulator, (henceforth to be abbreviated as MFP and SFP respectively), will shortly be discussed. Almost all the arithmetical and mathematical routines use the MFP and/or the SFP. The MFP occupies the memory locations $9D-$A2 and $AC. The exponent of the floating point number is in $9D (in excess 80 code), the mantissa is in $9E-$A1, and its sign is in $A2. Location $AC is used in most floating point routines as an extra mantissa byte, to increase the precision of the calculations. This location will further be called "the extension byte." An example of how one can convert the contents of the MFP to a decimal number is given in example 1 (page 31). The sign of the number is positive, since the first bit of $A2 is zero. In case this bit equals one, the sign of the number in the MFP will be negative. The exponent is calculated by converting the hex number 81 in $9D to decimal, which gives 129, and by subtracting the excess (=80 (hex) or 128 (decimal)) from it. The method that is used to convert the mantissa to decimal is essentially the same as the method used to convert a normal hex number to decimal, except that instead of the multiplicands 16, 256, 4,096, etc., the reciprocals of these numbers have to be used.

The number zero forms an exception to the rules mentioned above. Applesoft considers a number to be zero if the exponent ($9D) equals zero, independent of the value of the mantissa.

The results from arithmetical operations and mathematical functions in Applesoft are, in general, placed in the MFP. Next, the MFP is usually normalized and pushed on the stack or stored in memory. The normalizing of the MFP means that the bytes of the mantissa are rotated to the left (zeros enter at the right) until the left-most bit of $9E equals one. At every rotation the exponent is decreased by one, since rotating the mantissa one bit to the left means multiplying the number in the MFP by two, and this number must, of course, remain the same.

If, after the normalizing process, the MFP has to be stored in memory, it must be packed because the MFP occupies 7 bytes of memory, whereas Applesoft reserves only 5 bytes for the storage of real variables. In the packing routine, first the mantissa is rounded off by considering the left-most bit of the extension byte. If this bit equals one, the mantissa is increased by one, otherwise the mantissa remains the same. Then the sign is packed into the floating point number. If the sign is positive, the left-most bit of $9E is put equal to zero, otherwise it remains equal to one. Note that the sign can be packed in this way because the first bit of $9E contains no information since it always equals one after normalizing.

The SFP occupies the memory locations $A5-$AA. The exponent is in $A5, the mantissa in $A6-$A9, and its sign in $AA. The SFP has no extension byte. For the arithmetical and mathematical operations requiring two operands, the first operand has to be put in the MFP and the second operand in the SFP. Thus, loading the SFP and the MFP with two numbers and doing a JSR to, for instance, the multiplication routine, leaves the product of the numbers in the MFP. For some arithmetical routines it is necessary to determine—before the routine is executed—the exclusive OR of the signs of the numbers in the MFP and the SFP. The result must be stored in location $AB. This implies that the first bit of $AB must be one if the signs differ, otherwise the first bit has to equal zero. However, in most cases the user does not have to bother about determining the value of $AB, since it usually is not necessary to load the MFP and/or the SFP "by hand." Applesoft provides us with a lot of routines that can be used to get floating point numbers from memory, unpack them, and place them in the MFP or the SFP. All the routines that pull memory in the SFP also set $AB to the right value.

The Use of Applesoft Routines

The Applesoft subroutines that are, in my opinion, the most useful for m.l. programmers are listed in table 1. A distinction has been made between various types of subroutines, such as Copy, Errors, Conversion and Mathematical routines, etc. Rather than discussing each of the routines separately, a (very) simple example will be given to illustrate how to work with them. For a good understanding of this example, it is advisable to read the general remarks in table 1 first.

***Table 1:* Applesoft Routines (continued)**

$DE47: Pull stack in the SFP and determine $AB. This routine will usually be used in combination with subroutine $DE10. In that case it is for a successful execution of routine $DE47 necessary to push the return address of $DE47 on stack (high order byte first) before executing $DE10. Contrary to most other routines described here, $DE47 must be executed by means of a JMP instruction.

Exit(A = Z.F. = $9D, X = entry, Y = entry), modifies SFP,$AB.

4. SFP/MFP routines

$EB53: Copy SFP into MFP, put extension byte equal to zero.

Exit(A = $9D, X = 0, Y = entry), modifies MFP.

$EB63: Pack extension byte in MFP and copy MFP into SFP, put extension byte equal to zero.

Exit(A = $9D, X = 0, Y = entry), modifies MFP,SFP.

$EB66: Copy MFP (without extension byte) into SFP, put extension byte equal to zero.

Exit(A = $9D, X = 0, Y = entry), modifies MFP,SFP.

E. Errors

1. Purpose

If an error is detected in a m.l. program, one of the error routines may be used to print an error message.

2. Error messages

To print an error message, load the X register with the code of the message and execute a JMP to $D412 or execute a JMP to one of the locations listed behind the error messages. After printing the error message, Applesoft returns to BASIC command level (unless an ON ERR statement has been executed).

Code	Error message	JMP location
00	NEXT WITHOUT FOR	$DD0B
10	SYNTAX ERROR	$DEC9
16	RETURN WITHOUT GOSUB	$D979
2A	OUT OF DATA	—
35	ILLEGAL QUANTITY	$E199
45	OVERFLOW	$E8D5
4D	OUT OF MEMORY	$D410
5A	UNDEF'D STATEMENT	$D97C
6B	BAD SUBSCRIPT	$E196
78	REDIM'D ARRAY	—
85	DIVISION BY ZERO	$EAE1
95	ILLEGAL DIRECT	$E30B
A3	TYPE MISMATCH	$DD76
B0	STRING TOO LONG	—
BD	FORMULA TOO COMPLEX	$E430
D2	CAN'T CONTINUE	—
E0	UNDEF'D FUNCTION	$E30E

F. Expressions

1. Purpose

The Expressions routines can be used to evaluate expressions in an & statement. When calling an expression evaluation routine, the text pointer in $B8 and $B9 must point to the first character of the expression. After control is returned from the evaluation routine, the text pointer points to the first character behind the expression. In the evaluation routines below, this character is called the terminal sign. The terminal sign might, for instance, be a comma, but also a special character such as a "#". The locations that are modified by the routines are not specified here, since these depend on the type of the expression.

2. Expression evaluation routines

$DD67: Evaluate expression to next terminal sign, store result in MFP.

$E105: Evaluate expression to next terminal sign and convert result, which must be non-negative, to two byte integer in $A0 (high) and $A1 (low).

Exit(Y = 0).

$E6F8: Evaluate expression to next terminal sign and convert result, which must be non-negative, to a one byte integer in $A1.

Exit(A = terminal sign, X = $A1, Y = 0).

G. Init

1. Purpose

Initialize mantissa of the MFP or the SFP.

2. Initialization routines

$EC40: Init mantissa MFP (except extension byte) and Y to value in Accumulator.

Entry(A), Exit(A = entry, X = entry, Y = A), modifies MFP.

$E84E: Put MFP ($A2 and $9D) equal to zero.

Exit(A = Z.F. = 0, X = entry, Y = entry), modifies MFP.

H. Mathematical I (routines with one operand)

$EBAF: MFP = ABS(MFP)

Exit(A = entry, X = entry, Y = entry), modifies MFP.

$F09E: MFP = ATN(MFP)

Modifies $5E,$5F,$62-$66,$92-$9C,MFP,$A3,SFP,$AB,$AD,$AE.

$EED0: MFP = −MFP

Exit(X = entry, Y = entry), modifies MFP.

$EFEA: MFP = COS(MFP)

Modifies $D,$16,$5E,$5F,$62-$66,$92-$9C,MFP,$A3,SFP,$AB,$AD,$AE.

$EF09: MFP = EXP(MFP)

Modifies $D,$5E,$5F,$62-$65,$92,$98-$9C,MFP,$A3,SFP,$AB,$AD,$AE.

$EC23: MFP = INT(MFP)

Modifies $D,MFP.

$E941: MFP = LOG(MFP)

Modifies $5E,$5F,$62-$66,$92-$9C,MFP,$A3,SFP,$AB,$AD,$AE.

$DE98: MFP = NOT(MFP). This routine returns MFP = 1 if MFP = 0, else routine returns MFP = 0.

Modifies MFP, puts $11 equal to zero.

$EB90: MFP = SGN(MFP)

Exit(Y = 0), modifies MFP.

$EB82: Accumulator = SGN(MFP)

Exit(A = FF if MFP < 0, A = 0 if MFP = 0 and A = 1 if MFP > 0, X = entry, Y = entry).

$EFF1: MFP = SIN(MFP)

Modifies $D,$16,$5E,$5F,$62-$66,$92-$9C,MFP,$A3,SFP,$AB,$AD,$AE.

(continued)

Suppose one wishes to translate a BASIC subroutine to m.l. In that case the m.l. routine can be called from BASIC by means of the & symbol. The & symbol causes an unconditional jump to location $3F5 where the user can insert a JMP instruction to the start of the m.l. program.

After the execution of the & symbol, the text pointer of BASIC, which is in the locations $B8 and $B9, points to the next character of the line (spaces are ignored). Thus, if we have the line

```
10 & A1, BQ, C
```

where A1, BQ and C are reals, the text pointer points—after the execution of the & symbol—to the A. Suppose we wish to multiply A1 and BQ and store the result in C. We then first have to determine the starting location of the storage area of the value of A1 in memory. This can be done by making use of the subroutine $DFE3, listed under the heading Names in table 1. A JSR to $DFE3 in the m.l. program executes an Applesoft routine which puts the name of the variable (in this case A1) in $81 and $82; the status of the variable (in this case real) in $11 and $12; the pointer to the location of the variable in $9B and $9C; and, most important, the pointer to the value of the variable in $83 and $84, as well as in the Accumulator and the Y register.

Now that the starting location of the value of A1 is known, the value of A1 can be pulled into the MFP. For this purpose, the Copy routine $EAF9 can be used. Since the entry of this routine corresponds with the exit of $DFE3, the subroutine call to $EAF9 can be placed directly behind the subroutine call to $DFE3.

Now that we have stored A1 in the MFP, we can proceed to analyzing line 10. After the execution of subroutine $DFE3, the text pointer points to the first character behind the name of the variable, which is—in our example—a comma. If one plans to write a serious m.l. program, it might be useful to check whether there is indeed a comma behind the name.

For checking purposes, various routines are listed under the heading Charget-Charcheck. For instance, to check whether a comma is present, a JSR to $DEBE can be executed. In case the character is not a comma, the "SYNTAX ERROR" message is displayed and Applesoft gives a warm

start on BASIC. If, on the other hand, a comma is present, the text pointer is advanced and points now to the letter B.

To obtain the starting location of the storage area of the value of BQ, again a JSR $DFE3 is executed. Since A1 and BQ have to be multiplied, BQ must be stored in the SFP. To accomplish this, subroutine $E9E3 is used, which also can be placed directly behind the JSR $DFE3 instruction, because the exit of $DFE3 corresponds with the entry of $E9E3. Note that it is necessary to fill the MFP before the SFP, because $AB is set when BQ is pulled in the SFP.

As can be seen, the entry of the multiplication routine $E982 corresponds with the exit of $E9E3. Sc after the JSR to $E9E3, the multiplication can be carried out by means of a JSR $E982. Note that the m.l. program can be reduced by several bytes by using JSR $E97F instead of the last two mentioned subroutine calls.

Finally, the result of the multiplication, which is in the MFP, has to be stored in C. Before this is done, a JSR to $DEBE is executed to check whether the text pointer points to a comma. Next, the starting location of the storage area of the value of C is determined by means of a JSR $DFE3 instruction. To store the value of C in memory, the Copy routine $EB2B can be used. Since the entry of this routine is (X,Y), whereas the exit of $DFE3 is (A,Y), the instruction TAX must be inserted before the instruction JSR $EB2B.

After the last execution of $DFE3, the text pointer points to the end of line 10, so a RTS instruction in the m.l. program returns control to the BASIC program which will restart execution at the line number following line 10. The complete m.l. program is given in example 2 (page 40).

The routine $DFE3 can also be used to find the start of the storage area of integer (%) variables, elements of arrays, and arrays. If one wishes to use matrix expressions in the & statement, it is necessary to store the hex value 40 in $14 because otherwise Applesoft will interpret the matrix names in the & statement as names of simple variables. Be sure you don't forget to put $14 back on zero before returning to BASIC, because otherwise strange things may happen.

Table 1: Applesoft Routines (continued)

$EE8D: MFP = SQR(MFP)

Modifies $D,$5E,$5F,$62-$66,$8A-$8E,$92-$9C,MFP,$A3, SFP,$AB,$AD,$AE.

$F03A: MFP = TAN(MFP)

Modifies $D,$16,$5E,$5F,$62-$66,$8A-$8E,$92-$9C,MFP, $A3,SFP,$AB,$AD,$AE.

$EFAE: MFP = RND(MFP). See Applesoft manual for argument RND function.

Modifies $5E,$5F,$62-$65,$92,MFP,SFP,$C9-$CD.

I. Mathematical II (routines with two operands)

Add

$E7C1: MFP = SFP + MFP, $AB must be determined before subroutine call.

Entry(A = Z.F. = $9D), modifies $92,MFP,SFP.

$E7BE: Pull memory (→ A,Y) in SFP, determine $AB, add: MFP = SFP + MFP.

Entry(A,Y), modifies $5E,$5F,$92,MFP,SFP,$AB.

AND

$DF55: MFP = SFP AND MFP. Routine returns MFP = 1 if MFP and SFP are both unequal to zero, else routine returns MFP = 0.

Modifies MFP, puts $11 equal to zero.

Divide

$EA69: MFP = SFP/MFP, $AB must be determined before subroutine call.

Entry(A = Z.F. = $9D), modifies $62-$66,MFP,SFP.

$EA66: Pull memory (→ A,Y) in SFP, determine $AB, divide: MFP = SFP/MFP.

Entry(A,Y), modifies $5E,$5F,$62-$66,MFP,SFP,$AB.

Multiply

$E982: MFP = SFP × MFP, $AB must be determined before subroutine call.

Entry(A = Z.F. = $9D), modifies $62-$65,MFP.

$E97F: Pull memory (→ A,Y) in SFP, determine $AB, multiply: MFP = SFP × MFP.

Entry(A,Y), modifies $5E,$5F,$62-$65,MFP,SFP,$AB.

$E2B6: Multiply two byte integer in $AD (low) and $AE (high) with two byte integer in $64 (low) and Accumulator (high). Store product in X register (low) and Y register (high).

Entry(A), Exit(X = low byte product,Y = high byte product), modifies $65,$AD,$AE, puts $99 equal to zero.

Or

$DF4F: MFP = SFP OR MFP. Routine returns MFP = 0 if MFP = SFP = 0, else routine returns MFP = 1.

Modifies MFP, puts $11 equal to zero.

Power

$EE97: MFP = SFP^{MFP}.

Entry(A = Z.F. = $9D), modifies $D,$5E,$5F,$60-$66,$8A-$8E, $92-$9C,MFP,$A3,SFP,$AB,$AE,$AD.

Subtract

$E7AA: Determine $AB, subtract: MFP = SFP − MFP.
Modifies $92,MFP,SFP,$AB.

$E7A7: Pull memory (→ A,Y) in SFP, determine $AB, subtract: MFP = SFP − MFP.
Entry(A,Y), modifies $5E,$5F,$92,MFP,SFP,$AB.

J. Names

1. Purpose

The Names routine can be used—during the evaluation of the & statement—to find the name, the status and the starting location of the storage area of simple variables, array elements and arrays.

2. Name routine

$DFE3: At the start of the execution of $DFE3, the text pointer must point to the first character of the name. After the execution of $DFE3, the text pointer points to the first character behind the name and the name and status locations are filled according to the table below.

	Name variable or array (el.)		Status variable or array (el.)	
	$81	$82	$11	$12
Real	pos	pos	0	0
String ($)	pos	neg	FF	0
Integer (%)	neg	neg	0	80

For example, if a variable has the name AB, $81 and $82 will contain the hex values 41 and 42 respectively, whereas if a variable has the name AB%, $81 and $82 will be loaded with the hex values C1 and C2. In the latter case, $12 is put equal to the hex value 80, to indicate that the variable is integer valued.

Furthermore, Applesoft loads the pointer to the start of the storage area of the variable or the array in $9B (low) and $9C (high). The pointer to the start of the storage area of the value of the variable or the array element is loaded in A=$83 (low) and Y=$84 (high). If an array element is evaluated, the pointer to the first element of the array is stored in $94 (low) and $95 (high).

In case one wishes to use matrix expressions in the & statement (for instance & A = A − B, where A and B are matrices), the hex value 40 must be stored in $14 before executing $DFE3. Before returning to BASIC, $14 has to be reset to zero again.

Under the assumption that no strings are used in the BASIC program (which may lead to house cleaning activities), the following locations may be modified by the execution of $DFE3.

1. At the evaluation of simple variable names: $10, $11, $12, $81-$84, $94-$97, $9B, $9C, $B8, $B9.
2. At the evaluation of array elements: $F, $10-$12, $81-$84, $94-$97, $9B, $9C, MFP, $AE, $AD, $B8, $B9. In addition, other locations may be modified, depending on the expressions in the subscripts.
3. At the evaluation of (already dimensioned) array names which have to be interpreted as matrix names: $10, $11, $12, $81, $82, $9B, $9C, $B8, $B9.

(continued)

Apart from using names in the & statement, one can also insert expressions. For instance, & SIN(1) + SQR(B). To evaluate such an expression, subroutine $DD67 can be executed in the m.l. program. The result of the expression is stored in the MFP. If the result has to be converted to an integer value, a JSR $E105 or a JSR $E6F8 instruction can be used instead of the JSR $DD67 instruction.

It might be possible that a wrong input to the m.l. program, or an error during the execution of the m.l. program, is detected. This will, for example, be the case if a matrix that is to be inverted turns out to be not a square matrix. In that case, one may want to let Applesoft print an error message—indicating the kind of the error—with the line number of the & statement that caused the error. For this purpose, the routines listed under the heading Errors may be used. In the case of the wrongly dimensioned matrix, a JMP $E196 instruction, for instance, displays the message "BAD SUBSCRIPT IN XX." After displaying the message Applesoft returns to BASIC command level.

Although there are more routines in table 1, it seems superfluous to discuss them here, since it will now be obvious how to use them. Instead, an example will be given to show how to integrate some of the routines in a matrix multiplication program.

A Matrix Multiplication Program

When written in BASIC, a matrix multiplication subroutine consists of the statements

```
499 REM MATRIX MULTIPLICA-
    TION : C(R,P) = A(R,S) ×
    B(S,P)
500 FOR I = 1 TO P
510 FOR J = 1 TO R
520 LET D = 0
530 FOR K = 1 TO S
540 LET D = D + A(J,K) ×
    B(K,I)
550 NEXT K
560 LET C(J,I) = D
570 NEXT J
580 NEXT I
590 RETURN
```

To execute this subroutine, the following main program can be used:

```
 10 INPUT "DIMENSIONS
    MATRICES P,R,S ? ";P,R,S
 20 DIM A(R,S),B(S,P),C(R,P)
 30 FOR I = 1 TO R
 40 FOR J = 1 TO S
 50 LET A(I,J) = I + J
 60 NEXT J
 70 NEXT I
 80 FOR I = 1 TO S
 90 FOR J = 1 TO P
100 LET B(I,J) = I × J
110 NEXT J
120 NEXT I
130 GOSUB 500
140 STOP
```

In the main program, the matrices A and B are dimensioned and initialized. Next, the matrix multiplication subroutine is called to put C equal to the product of A and B. If a m.l. program is written to multiply two matrices, the subroutine call at line 130 can be replaced by

```
130 & C = A × B
```

Although the matrix names in the & statement can be chosen freely, we will use in the sequel the names C, A and B to denote the respective matrices. The dimensions of the matrices will be denoted by the same letters as in the BASIC program (i.e. P, R and S).

The m.l. program can be split up into a main program and several subroutines. The main program performs the evaluation of the & statement. The first subroutine, called FNAME, takes care of the calculation of the pointers to the storage areas of the matrices C, A, and B in memory. The second subroutine, called MATMULT, is used for the actual matrix multiplication. Two other subroutines, ADD and ADD5, are called by FNAME and MATMULT to do some frequently occuring additions. A discussion of the functions of the various routines—which are listed in table 2— follows.

1) The main program ($4000-$4022)

The main program is written solely to control the multiplication of two matrices. Therefore it has to be replaced by another main program if the number of matrix operations is extended (with, for instance, add, subtract and inverse). The comment inserted in the listing shows how the program works.

***Table 1:* Applesoft Routines (continued)**

K. Normalize

$E82E: Normalize MFP
Exit(Y=0), modifies MFP.

L. Pack

$EB72: Pack extension byte in MFP.
Exit(X=entry,Y=entry), modifies MFP.

Example 2

$3F5 : JMP $5000	Jump to multiplication program
$5000 : JSR $DFE3	Find starting location of value of first variable
$5003 : JSR $EAF9	Pull first variable into the MFP
$5006 : JSR $DEBE	Check on comma in & statement
$5009 : JSR $DFE3	Find starting location of value of second variable
$500C : JSR $E9E3] JSR $E97F	Pull second variable in the SFP, and
$500F : JSR $E982]	Multiply MFP with SFP, store product in MFP
$5012 : JSR $DEBE	Check on comma
$5015 : JSR $DFE3	Find starting location of value of third variable
$5018 : TAX	Prepare entry store routine
$5019 : JSR $EB2B	Store product in third variable
$501C : RTS	Return to BASIC

***Table 2:* Listings of Machine Language Programs**

A. The main program

Purpose

The evaluation of the & statement: & C = A × B, where C,A and B are matrices.

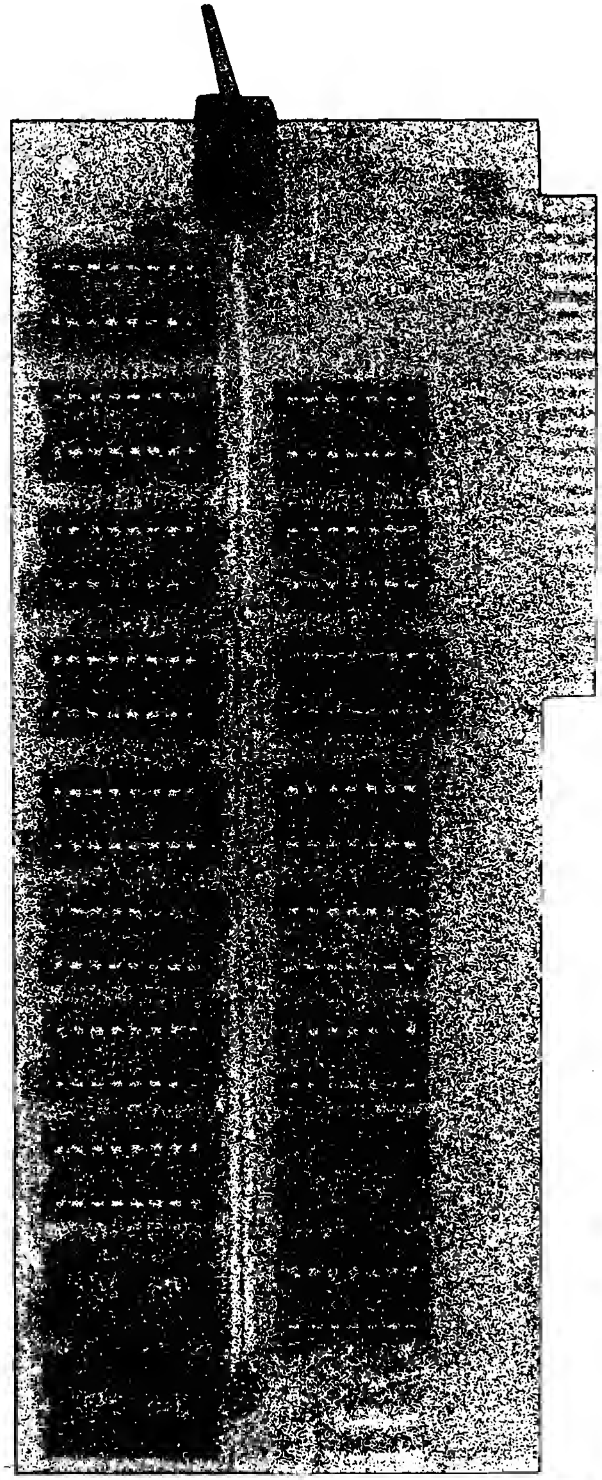
Listing	Comment
$3F5 : JMP $4000	Init jump location for & statement.
$4000 : LDX #$F8 $4002 : STX $08	Init location $08 for FNAME.
$4004 : JSR $4025	Execute FNAME on first matrix (C).
$4007 : LDA #$D0 $4009 : JSR $DEC0	Check on "=" in & statement.
$400C : JSR $4025	Execute FNAME on second matrix (A).
$400F : LDA #$CA $4011 : JSR $DEC0	Check on "×" in & statement.
$4014 : JSR $4025	Execute FNAME on third matrix (B).
$4017 : LDA $06 $4019 : STA $71 $401B : LDA $07 $401D : STA $72	Restore column length of A (=column length of C) in $71 and $72.
$401F : JSR $40A5	Execute MATMULT.
$4022 : RTS	Return to BASIC.

B. Subroutine FNAME

Purpose

Find name of array, check whether array has two dimensions, each less than 256. Store dimensions in $FC,X (second dimension) and $FD,X (first dimension). Calculate column length of array (in bytes) and store it in $71 (low) and $72 (high). Calculate pointer to storage area of first element of second column of array, and store pointer in $0,X+2 and $1,X+2. FNAME can be called successively three times (or less). Before the first call, the hex value F8 must be stored in location $08. At the start of FNAME, the X register is loaded with the value in location $08. During the execution of FNAME the X register is incremented by two and stored in location $08 so that the contents of location $08 are incremented by two each time FNAME is called.

(continued)



2) Subroutine FNAME ($4025-$4083)

Contrary to the main program, FNAME is constructed in such a way that it can be used for other matrix operations too. The main purpose of FNAME is to calculate the pointer to the first element of the second column of the array being evaluated. The second column is taken because it is customary to use—when working with matrices—non zero values of the subscripts only, whereas Applesoft reserves—when it encounters a DIM X(P,R) statement—P + 1 rows and R + 1 columns for the array because it allows zero subscripts. As an example, suppose that a DIM X(2,3) instruction is executed in a BASIC program. Applesoft stores the X-array columnwise (example 3, page 46).

When multiplying the X matrix with another matrix, only the underscored elements have to be taken into account. Since the first column contains no underscored elements, it can be skipped.

Subroutine FNAME can be called successively (at most) three times. At the first call, location $08 must contain the hex value F8, being the start of the storage area, plus 4 of the matrix information. Consulting the memory map of FNAME in table 2, it can be seen that the dimensions of the C array, P + 1 and R + 1, are stored in $F4 and $F5 and the pointer to the first element of the second column of the C array (i.e., the pointer to C(0,1) in $FA and $FB. Since location $08 is automatically incremented by 2, each time FNAME is called, the dimensions of the next array (i.e., the A array) will—at the second call of FNAME—be stored in $F6 and $F7 and the pointer in $FC and $FD. The information of the B array is stored in $F8, $F9, $FE and $FF.

Apart from the calculation of the pointer, FNAME also checks whether the array being evaluated has two dimensions, and whether the size of each dimension is less than 256. The latter check is necessary because MATMULT can handle matrices with dimensions less than 255 only, which will be sufficient for almost all practical purposes.

Finally, at each call of FNAME, the column length of the array being evaluated (which equals 5 times the number of column elements, since reals use 5 bytes of memory) is calculated and stored in $71 and $72.

***Table 2:* Listings of Machine Language Programs (continued)**

Memory Map of FNAME

$06 $07	Column length (in bytes) of first array (C).
$08	Pointer to storage area of array information.
$71 $72	Column length (in bytes) of third array (B).
$F4	Second dimension of first array (C).
$F5	First dimension of first array (C).
$F6 $F7	Idem for second array (A).
$F8 $F9	Idem for third array (B)
$FA $FB	Pointer to first element of second column of first array (C).
$FC $FD	Idem for second array (A).
$FE $FF	Idem for third array (B).

Listing FNAME	Comment
$4025 : LDA #$40 $4027 : STA $14	Put $14 equal to 40 for search of matrix name.
$4029 : JSR $DFE3	Determine pointer to start storage area of array.
$402C : LDX $08	X is loaded with pointer to storage area array information.
$402E : LDA $12 $4030 : ORA $11 $4032 : BEQ $4037	Check whether array contains reals.
$4034 : JMP $DD76	If not, display "TYPE MISMATCH".
$4037 : STA $14	Put $14 back on zero.
$4039 : LDY #$04	
$403B : LDA #$02	
$403D : CMP ($9B),Y	Compare number of dimensions of array with 2.
$403F : BEQ $4044	
$4041 : JMP $E196	If not equal, display "BAD SUBSCRIPT".
$4044 : INY	
$4045 : LDA($9B),Y $4047 : BNE $4041 $4049 : INY $404A : LDA($9B),Y $404C : STA $FC,X $404E : INX $404F : INY $4050 : CPY #$09 $4052 : BNE $4045	Check whether dimensions of array are both less than 256. If not, display "BAD SUBSCRIPT". If yes, store second dimension in $FC,X and first dimension in $FD,X. Note that at the first call of FNAME, the X register is loaded with F8. The dimensions of the first array are thus stored in $F4 and $F5.
$4054 : TYA	Accumulator contains 9 here.
$4055 : CLC $4056 : ADC $9B $4058 : STA $00,X $405A : LDA $9C $405C : ADC #$00 $405E : STA $01,X	To obtain the pointer to the storage area of the first element in the array, 9 is added to the pointer in $9B and $9C. The result is stored in $00,X (low) and $01,X (high). At the first call of FNAME X equals FA here, so the pointer is stored in $FA and $FB.
$4060 : LDA $FB,X $4062 : LDY #$00 $4064 : STY $72 $4066 : ASL $4067 : ROL $72 $4069 : ASL $406A : ROL $72 $406C : ADC $FB,X $406E : STA $71 $4070 : BCC $4074 $4072 : INC $72	Calculate the column length of the array by multiplying the size of the first dimension (which was stored in $FB,X) with 5. The column length is stored in $71 and $72.

```
$4074 : CPX #$FA     Is it the first call of FNAME?
$4076 : BNE $407E
$4078 : STA $06    ] If yes, save column length of the array in $06 and
$407A : LDY $72    ] $07.
$407C : STY $07    ]
$407E : JSR $4087    Add column length to the last calculated pointer
$4081 : STX $08      to obtain the pointer to the storage area of the
$4083 : RTS          first element of the second column of the array.
                     Store X in $08 for next calls of FNAME and
                     return to main program.
```

C. Subroutines ADD and ADD5

Purpose ADD

Add two byte integer in $71 (low) and $72 (high) to two byte integer in $00,X (low) and $01,X (high). Store result in $00,X (low) and $01,X (high).

Entry(X), Exit(A = $01,X,X = entry,Y = entry).

Purpose ADD5

ADD 5 to two byte integer in $00,X (low) and $01,X (high). Store result in $00,X (low) and $01,X (high).

Entry(X), Exit(A = $00,X,X = entry,Y = entry).

Listing ADD	**Listing ADD5**
$4085 : LDA $71	$4095 : CLC
$4087 : CLC	$4096 : LDA $00,X
$4088 : ADC $00,X	$4098 : ADC #$05
$408A : STA $00,X	$409A : STA $00,X
$408C : LDA $72	$409C : BCC $40A0
$408E : ADC $01,X	$409E : INC $01,X
$4090 : STA $01,X	$40A0 : RTS
$4092 : RTS	

D. Subroutine MATMULT

Purpose

Multiply matrix A(R,S)	(dimensions in $F6 (S+1) and $F7 (r+1), pointer in $FC and $FD)
with matrix B(S,P)	(dimensions in $F8 (P+1) and $F9 (S+1), pointer in $FE and $FF)
and store result in matrix C(R,P)	(dimensions in $F4 (P+1) and $F5 (R+1), pointer in $FA and $FB)

where P,R, and S each have to be less than 255.

Memory map of MATMULT

$06	cr : Row counter for C.
$07	cs : Multiplication counter for row/column multiplication.
$17, $18	hp_A : pointer to first element of current row of A.
$19, $1A	p_B : pointer to first element of current column of B.
$71, $72	k = 5(R+1) : Column length of A (in memory).
$F4	P + 1 at entry. Used as column counter for C.
$F5	R + 1 = number of elements per column of C (in memory)
$F6	S + 1 : S equals the number of multiplications necessary to multiply a row of A with a column of B.
$F8, $F9	pc_A : pointer to current element of A.
$FA, $FB	pc_C : pointer to current element of C.
$FC, $FD	p_A : pointer to first element of second column of A.
$FE, $FF	pc_B : pointer to current element of B.

(continued)

The column length of the C array, which equals the column length of the A array, is saved in locations $06 and $07, because the latter column length is needed later for MATMULT.

3) Subroutine MATMULT ($40A5-$4124)

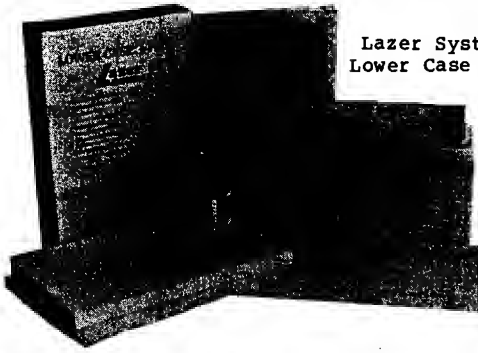
Before the matrices are multiplied, the dimensions are checked to determine whether they satisfy the conditions for multiplication. Next, the multiplication is carried out as indicated by the flow diagram in figure 1. The flow diagram shows that the pth column of C (p = 2,..P+1) is obtained by multiplying the R rows of A (i.e., row 2,...R+1) each with the pth column of B (p = 2,...P+1). Note that at a row/column multiplication, the product of the first element of a row and the first element of a column is omitted since these elements have zero subscripts. The elements of the rows of A are separated by a distance of k (= 5(R+1)) bytes from each other in memory, so that each time a next row element of A is needed, k has to be added to pc_A. After a row of A, not being the last row, has been multiplied with a column of B, the hp_A pointer is incremented by 5 and pc_A is put equal to hp_A, so that pc_A now points to the second element of the next row of A. If a column of C, not being the last column, has been filled, hp_A is put equal to its starting value. That is, p_A, and p_B is put equal to pc_B, which at that time points to the first element of the next column of B.

The flow diagram further shows how the multiplication of a row with a column is performed. The stack is used to store the sum of the products obtained so far, and each time a row element is multiplied with a column element, the stack is pulled into the SFP and the newly-obtained product is added to it. The result is then pushed on the stack again. This process is continued until the row/column multiplication is ready. The row/column product is then stored in memory.

Note that the subroutine address pushed on the stack at locations $40CE-$40D3 is the address minus one of the instruction following the JMP instruction at location $40E5. If the MATMULT subroutine is relocated to another part of memory, this subroutine address must be adjusted.

LOWER CASE +PLUS By Lazer SYSTEMS
for the Apple II Computer

Guess which Apple (tm) compatible lower case adapter costs less :



Lazer Systems' Lower Case +Plus

Dan Paymar's LCA-1

(You would probably pick the one on the bottom becuase you can see that you get well over twice as much in the package on top.)

WRONG!! Both the Lazer Systems' Lower Case +Plus (tm) (on the top) and the Dan Paymar LCA-1 (tm) (on the bottom) cost $59.95 (Paymar lowered the price of his Lower Case adapter from $64.95 shortly after the release of the Lower Case +Plus). Even if the Lower Case +Plus was priced at $79.95, it would still be a better buy.

The Lower Case +Plus uses a high quality double sided printed circuit board with solder mask and silkscreen. The Paymar LCA-1 uses a low-cost single sided board. The Lower Case +Plus uses a 2716-compatible ROM for its character generator. You can readily create your own character sets using a HI-RES character generator, such as the "Keyboard Filter", sold with the ROMPLUS from Mountain Computer. The LCA-1 uses a fixed, non-modifiable character set. The Lower Case +Plus comes with two character sets, while the LCA-1 has only one. The Lower Case +Plus supports 128 displayable characters, while the LCA-1 supports only 96.

Word Processor Compatibility

The Dan Paymar LCA-1 made word processing on the Apple II practical; Lazer Systems made it even better. The Lazer Systems Lower Case +Plus is compatible with all word processing programs which can utilize the Paymar LCA-1 for lower case display.

One Board Works with all Apples

Apple Computer recently changed the design of their character generator logic rendering the LCA-1 useless in the newer Apple II's. So, Dan Paymar created the LCA-2 which works only with the newer Apples. There's only one problem, you have to know which Apple you own before ordering. This problem is nonexistant with the Lazer Systems' Lower Case +Plus because our board works with both the older and newer Apples.

A Complete Package

For $59.95 Dan Paymar will sell you an LCA-1 ($49.95 for the LCA-2 EPROM), some descriptive literature and a plastic bag. (see the photo)

For $59.95 Lazer Systems provides you with the Lower Case +Plus, Basic software on disk and over 40 pages of user documentation all neatly packaged (see photo for entire contents*). Pascal and Applewriter patches are provided with our documentation. Pascal users may elect to purchase Pascal software on diskette for $9.95. Both the Basic and Pascal software packages give you the capability of entering all 96 printable characters into your programs.

The Expansion Socket (Our Exclusive)

No one else has it. Not Paymar, Videx, Data shifter (Muse) nor Uni-Text. The expansion socket on the Lazer Systems' Lower Case +Plus gives you capabilities found only on the Apple III. By adding our Graphics +Plus (tm) your Apple can software select either of the two character sets on the Lower Case +Plus or a third, RAM-based, character set on the Graphics +Plus. With a RAM-based character set, you can program high resolution graphics animation which will run up to ten times faster than equivalent programs using the Apple's HI-RES graphics and requires only one eighth the memory.

The other addition to our expanding line is the Lazer Systems' Keyboard +Plus (tm). This device allows you to enter upper and lower case using the shift key on your Apple keyboard. Imagine what this feature will do for word processing applications. The Keyboard +Plus also incorporates a FIFO buffer which allows you to continue typing even though the computer is busy. Have you ever lost characters because the Apple only retains the last character typed? With a Keyboard +Plus installed you won't have to wait for your computer.

The Keyboard +Plus and Graphics +Plus will be premiered and available at the West Coast Computer Fair in San Francisco.

A Special Offer to Lessen the Sting

If you are considering the purchase of a Lower Case adapter, we're sure you will feel the Lazer Systems' Lower Case +Plus is your best buy. If you already own a Dan Paymar LCA, you may want to switch to the Lower Case +Plus. So, for those of you who already own a Dan Paymar LCA-1, we will grant you an $18.00 trade-in allowance towards the purchase of a Lower Case +Plus. Sorry, for the LCA-2, owners we can offer only $10.00 since it is nothing more than a 2716 EPROM with useless data that will have to be cleaned before the EPROM can be used again. This offer expires April 30, 1981.

ORDER FROM: Lazer SYSTEMS
P.O.Box 55518
Riverside, Calif. 92517
(714) 682-5268

We gladly accept Mastercard and Visa. Include card#, expiration date and signature.

Lower Case +Plus.	$59.95
Pascal Software	$9.95

Add $2.00 shipping & handling to all orders.
Calif. residents add 6% sales tax.
Outside U.S.A. requires additional charges:
Canada & Mexico add $7.00
All other countries add $15.00.
Foreign orders must be pre-paid by Mastercard, Visa or certified check in U.S. dollars.
The information presented in this ad was accurate at the time of writing, December 26, 1980.
*Three ICs shown plugged in comes from your computer.
Dealer inquiries invited.

***Table 2:* Listings of Machine Language Programs (continued)**

Listing MATMULT	Comment
$40A5 : LDA $F4	
$40A7 : CMP $F8	
$40A9 : BEQ $40AE	
$40AB : JMP $E196	Check dimensions for multiplication. If an error is detected, display "BAD SUBSCRIPT".
$40AE : LDA $F5	
$40B0 : CMP $F7	
$40B2 : BNE $40AB	
$40B4 : LDA $F6	
$40B6 : CMP $F9	
$40B8 : BNE $40AB	
$40BA : DEC $F4	P = P − 1 : decrement column counter for C.
$40BC : BNE $40BF	If P equals zero, matrix multiplication is ready
$40BE : RTS	Return to main program.
$40BF : LDA $F5	cr = R + 1 : init row counter for C.
$40C1 : STA $06	
$40C3 : LDX #$03	
$40C5 : LDA $FC,X	
$40C7 : STA $17,X	$hp_A = p_A$
$40C9 : DEX	$p_B = cp_B$
$40CA : BPL $40C5	
$40CC : BMI $4100	Always taken.
$40CE : LDA #$40	
$40D0 : PHA	Push subroutine address for routine $DE47 on stack.
$40D1 : LDA #$E7	
$40D3 : PHA	
$40D4 : JSR $DE10	Push MFP on stack.
$40D7 : LDA $F8	
$40D9 : LDY $F9	Load (pc_A) in MFP.
$40DB : JSR $EAF9	
$40DE : LDA $FE	
$40E0 : LDY $FF	Load (pc_B) in SFP and
$40E2 : JSR $E97F	Multiply MFP with SFP. Store product in MFP.
$40E5 : JMP $DE47	Pull stack into SFP.
$40E8 : JSR $E7C1	Add MFP and SFP. Store sum in MFP.
$40EB : LDX #$F8	$pc_A = pc_A + k$
$40ED : JSR $4085	
$40F0 : LDX #$FE	$pc_B = pc_B + 5$
$40F2 : JSR $4095	
$40F5 : DEC $07	cs = cs − 1
$40F7 : BNE $40CE	If cs equals zero, row/column product is ready.
$40F9 : LDX $FA	
$40FB : LDY $FB	Store MFP in (pc_C).
$40FD : JSR $EB2B	
$4100 : LDX #$FA	$pc_C = pc_C + 5$
$4102 : JSR $4095	
$4105 : DEC $06	cr = cr − 1
$4107 : BEQ $40BA	Is column of C filled? If yes, init hp_A, p_B and cr.
$4109 : LDX #$17	$hp_A = hp_A + 5$
$410B : JSR $4095	
$410E : STA $F8	
$4110 : LDA $18	$pc_A = hp_A$
$4112 : STA $F9	
$4114 : LDA $19	
$4116 : STA $FE	
$4118 : LDA $1A	$pc_B = p_B$: restore column counter for B.
$411A : STA $FF	
$411C : LDA $F6	
$411E : STA $07	cs = S + 1 : init multiplication counter.
$4120 : JSR $E84E	Initialize MFP to zero.
$4123 : BEQ $40F0	Always taken.

Some Final Remarks

Probably not all Apple owners will have Applesoft in ROM. However, since the disk and tape versions of Applesoft which I have seen do not differ by more than a few bytes from the ROM version, it will be no big problem to convert the entries of the routines listed in table 1 to these versions. The easiest way to do this is to find someone who has Applesoft in ROM, so that the differences can readily be traced back by comparing the versions with each other. In case no ROM version is available, one can use the subroutine entry locations, which are found at the beginning of the Applesoft program. The sequence of the first 64 subroutine entry locations* corresponds with the listing of the tokens in the Applesoft manual (#A2L0006 on page 121). Next, the entry locations of the routines for SIGN to MID$ follow. The rest of the entry locations* are for +, −, ×, /, O , AND, OR, unary minus, NOT and comparison. Before each of the latter entries, a code, indicating the order of the operation, is inserted.

Looking at table 3, where the entries of the routines for the ROM version are listed alphabetically, with the entries found in tape or disk versions of Applesoft, it will become apparent what differences there are. After that, the entry locations of the routines in table 1 can be converted accordingly.

As a last point I wish to express my admiration for the ingenuity of the writers of Applesoft. During my study of Applesoft, I often searched for hours for what was happening, in a seemingly endless sequence of (recursive) subroutines, which taught me a lot about m.l. programming. Apart from a few errors that were made (for instance, a zero byte forgotten between $E101 and $E102, so that the program

```
10 A% = −32768.00049 :
   PRINT A%
```

gives a surprising result) I came to the conclusion that Applesoft is a very good interpreter.

I also wish to express my gratitude to Mr. F. Curvers of the Erasmus University in Rotterdam, for providing me with an excellent cross reference of Applesoft, without which my work would have been far more difficult.

---

*Add one to the location found in the listing because the subroutines are executed via the RTS instruction.

Cornelis Bongers is an assistant professor of statistics at the Erasmus University in Rotterdam. He uses his Apple II for solving statistical problems (for instance, likelihood maximalization). Another important field of application is finding the solution of standardization problems, which in essence means: finding the set of sizes that minimizes the overall costs caused by the standardization of a product. As a hobby, he develops utility programs for the Apple, such as an assembler cross reference program and a disk-to-tape dump utility.

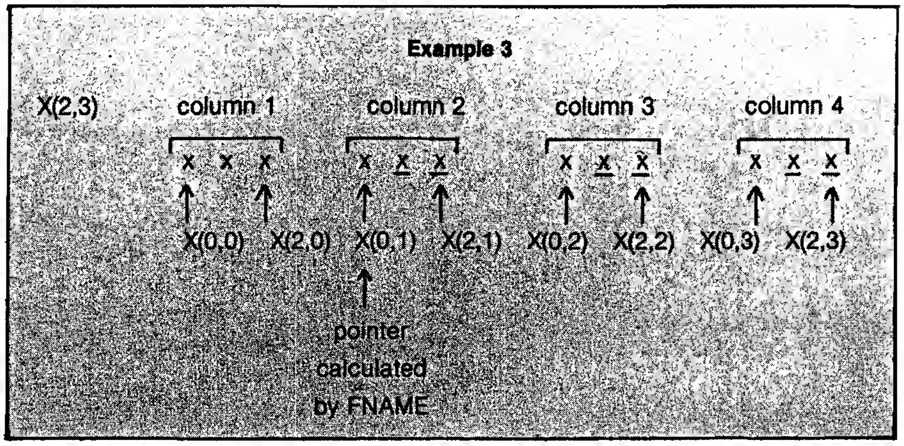


***Table 3:* Applesoft (ROM) Entry Locations**

Entry	Dec token	Key-word	Entry	Dec token	Key-word	Entry	Dec token	Key-word
$3F5	175	&	$F3D8	144	HGR2	$F3BC	167	RECALL
$E982	202	×	$F286	163	HIMEM:	$D9DC	178	REM
$E7C1	200	+	$F232	142	HLIN	$D849	174	RESTORE
$E7AA	201	–	$FC58	151	HOME	$F318	166	RESUME
$EA69	203	/	$F6FE	147	HPLOT	$D96B	177	RETURN
	209	<	$F7E7	150	HTAB	$E686	233	RIGHT$
$DF65	208	=	$D9C9	173	IF	$EFAE	219	RND
	207	>	$F1DE	139	IN#	$F721	152	ROT=
$EBAF	212	ABS	$DBB2	132	INPUT	$D912	172	RUN
$DF55	205	AND	$EC23	211	INT	$D8B0	183	SAVE
$E6E5	230	ASC	$F277	158	INVERSE	$F727	153	SCALE=
—	197	AT	$E65A	232	LEFT$	$DEF9	215	SCRN(
$F09E	225	ATN	$E6D6	227	LEN	$EB90	210	SGN
$F1D5	140	CALL	$DA46	170	LET	$F775	154	SHLOAD
$E646	231	CHR$	$D6A5	188	LIST	$EFF1	223	SIN
$D66A	189	CLEAR	$D8C9	182	LOAD	$DB16	195	SPC(
$F24F	160	COLOR=	$E941	220	LOG	$F262	169	SPEED=
$D896	187	CONT	$F2A6	164	LOMEM:	$EE8D	218	SQR
$EFEA	222	COS	$E691	234	MID$	—	199	STEP
$D995	131	DATA	$D649	191	NEW	$D86E	179	STOP
$E313	184	DEF	$DCF9	130	NEXT	$F39F	168	STORE
$F331	133	DEL	$F273	157	NORMAL	$E3C5	228	STR$
$DFD9	134	DIM	$DE98	198	NOT	$DB16	192	TAB(
$F769	148	DRAW	$F26F	156	NOTRACE	$F03A	224	TAN
$D870	128	END	$D9EC	180	ON	$F399	137	TEXT
$EF09	221	EXP	$F2CB	165	ONERR	—	196	THEN
$F280	159	FLASH	$DF4F	206	OR	—	193	TO
$E354	194	FN	$DFCD	216	PDL	$F26D	155	TRACE
$D766	129	FOR	$E764	226	PEEK	$A	213	USR
$E2DE	214	FRE	$F225	141	PLOT	$E707	229	VAL
$DBA0	190	GET	$E77B	185	POKE	$F241	143	VLIN
$D921	176	GOSUB	$D96B	161	POP	$F256	162	VTAB
$D93E	171	GOTO	$E2FF	217	POS	$E784	181	WAIT
$F390	136	GR	$F1E5	138	PR#	$F76F	149	XDRAW
$F6E9	146	HCOLOR=	$DAD5	186	PRINT	$EE97	204	^
$F3E2	145	HGR	$DBE2	135	READ			

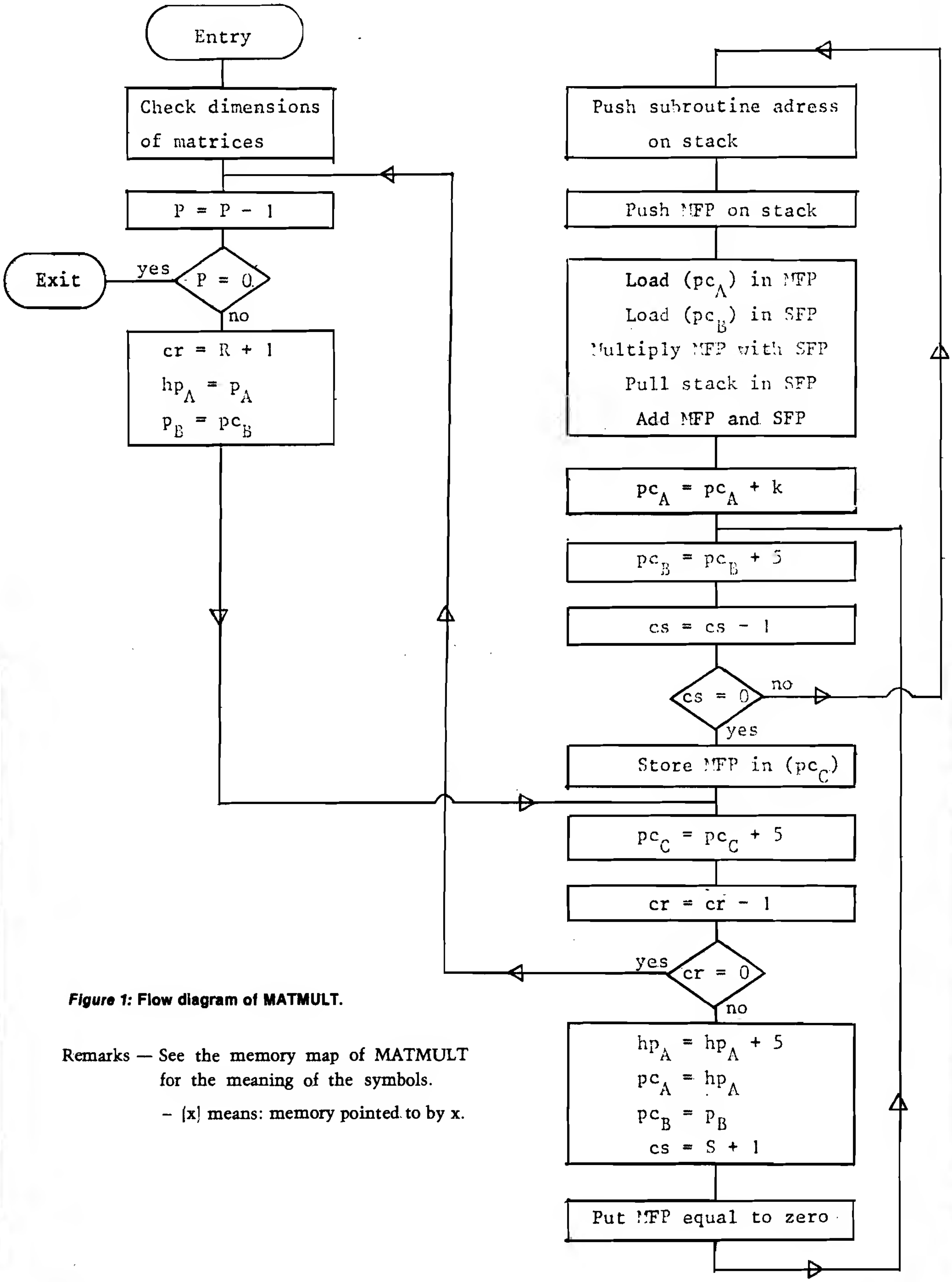


***Figure 1:* Flow diagram of MATMULT.**

Remarks — See the memory map of MATMULT for the meaning of the symbols.
– (x) means: memory pointed to by x.

One Dimensional Life on the AIM 65

This adaptation for the AIM 65 of Millen's "One-Dimensional Life," takes advantage of the AIM's 20-character LED display and its thermal printer.

Larry Kollar
257 W. Wadsworth Hall, MTU
Houghton, Michigan 49931

Thumbing through some back issues of *BYTE*, I came across Jonathan K. Millen's "One-Dimensional Life" (December 1978, pgs. 68-74). This article was particularly interesting, since I own a Rockwell AIM 65 microcomputer and I had been planning to write a two-dimensional Life game which would display each generation on the little printer. Dr. Millen's idea looked much easier to write and would certainly use less paper, since I could simply use the AIM's 20-character alphanumeric display.

Listing 1 is an assembly language listing of the One-Dimensional Life program. The program uses a 20 cell universe, with spaces outside this area being barren. My algorithm differs somewhat from that suggested by Dr. Millen in that two complete arrays are used in calculating each generation; Dr. Millen used a temporary storage space three cells in width, which follows the line of march as each cell is recomputed. I decided that I could write a simpler program by using the extra array.

Listing 1

```
0200                    ORG   $0200
0200 A9 20              LDA   #$20     FILL WORK AND
0202 A2 2B              LDX   #$2B     DISPLAY ARRAYS
0204 95 00     LBLA     STA   $0000,X  WITH
0206 CA                 DEX
0207 10 FB              BPL   LBLA     ASCII SPACES
0209 A2 00              LDX   #$00     CLEAR X
020B 20 93 E9  LBLB     JSR   INALL    GET A CHARACTER
020E C9 0D              CMP   #$0D     CARRIAGE RETURN?
0210 F0 0F              BEQ   LBLD     YES--START LOOPING
0212 C9 2A              CMP   #$2A     IS IT A '*'?
0214 F0 02              BEQ   LBLC     YES--STORE IT
0216 A9 20              LDA   #$20     NO--ASSUME A SPACE
0218 95 00     LBLC     STA   $0000,X  AND STORE IT
021A 95 16              STA   $0016,X  IN BOTH ARRAYS
021C E8                 INX            GO TILL 20 CHARS
021D E0 14              CPX   $#14     DONE YET?
021F 30 EA              BMI   LBLB     NO--GET MORE CHARS
0221 A2 14     LBLD     LDX   #$14     START COMPUTING GEN.
0223 18        LBLE     CLC            SETUP FOR ADDITION
0224 B5 14              LDA   $0014,X  ADD BOTH
0226 75 15              ADC   $0015,X  BEFORE
0228 75 16              ADC   $0016,X  THE MIDDLE,
022A 75 17              ADC   $0017,X  AND THE TWO
022C 75 18              ADC   $0018,X  FOLLOWING
022E 48                 PHA            AND SAVE IT
022F B5 16              LDA   $0016,X  SEE WHAT WE HAVE
0231 C9 20              CMP   #$20     IS IT A SPACE?
0233 D0 14              BNE   LBLI     NO--THE CELL IS LIVE
0235 68                 PLA            TEST DEAD CELLS
0236 C9 B4              CMP   #$B4     2 NEIGHBORS?
0238 F0 0B              BEQ   LBLH     YES...
023A C9 BE              CMP   #$BE     3 NEIGHBORS?
023C F0 0B              BEQ   LBLI     YES...
023E A9 20     LBLF     LDA   #$20     NO--IT REMAINS DEAD
0240 95 00     LBLG     STA   $0000,X  PUT IN DISPLAY AREA
0242 4C 54 02           JMP   LBLJ     AND GO FOR MORE
0245 A9 2A     LBLH     LDA   #$2A     BIRTH....
0247 D0 F7              BNE   LBLG
0249 68        LBLI     PLA            LIVING CELLS--TEST
024A C9 BE              CMP   #$BE     2 NEIGHBORS?
024C F0 F7              BEQ   LBLH     YES, IT SURVIVES
024E C9 D2              CMP   #$D2     4 NEIGHBORS?
0250 F0 F3              BEQ   LBLH     YES....
0252 D0 EA              BNE   LBLF     NO--THIS ONE DIES
```

(continued)

To run the program, type [* = 200], [G], (space). Set up the first generation by using asterisks for living cells; spaces, or any other characters, will be considered dead and will be entered as spaces. The program will begin executing if RETURN is hit or 20 cells have been typed in. If the program is stopped, it can be restarted at [* = 0221] and it will continue executing. See table 1 for other locations.

Comments

Obviously, a 20 cell universe is going to have some restrictions. The pattern shown in Photo 1 of Dr. Millen's article runs out of space on my display after the last generation shown on the video screen. Many other patterns quickly run out of space. But the program gives one a good flavor for what can and cannot be done in One-Dimensional Life, and there is enough room for discoveries, such as a period 3 glider eating the period 1 glider (see figure 1).

Also, the boundaries may help some patterns stabilize more quickly. Filling the entire display with living cells yields an oscillator with period 12. Gliders, no matter what stage they are in when they hit the edge, become flip-flops or oscillating patterns. Some oscillators will continue undisturbed when close enough to the edge for a part to extend into barren territory, others will mutate into blinkers or flip-flops.

Like conventional Life patterns, One-Dimensional Life patterns tend to mutate to more symmetrical patterns. However, one-dimensional patterns show a tendency to fight harder to survive than their conventional counterparts.

Figures 2 - 4 show some other results.

Listing 1 (continued)

```
0254 CA          LBLJ    DEX                 GENERATION FINISHED?
0255 10 CC               BPL      LBLE       NO--CONTINUE CHECKS
0257 20 F0 E9            JSR      CRLF       OUTPUT CR & LF
025A A2 00               LDX      #$00       DISPLAY GENERATION
025C B5 00       LBLK    LDA      $0000,X    GET NEW GENERATION
025E 95 16               STA      $0016,X    COPY INTO WORK ARRAY
0260 20 BC E9            JSR      OUTALL     AND DISPLAY IT
0263 E8                  INX
0264 E0 14               CPX      #$14       UNTIL ALL FINISHED,
0266 30 F4               BMI      LBLK       KEEP GOING
0268 A0 00               LDY      #$00       DELAY LOOP--CLEAR Y
026A A9 71       LBLL    LDA      #$71       SET UP FOR
026C 8D 08 A8            STA      TTWOL      A 3/4 SECOND
026F A9 0B               LDA      #$0B       DELAY...
0271 8D 09 A8            STA      TTWOH
0274 20 1B EC            JSR      TIMER      CALL INNER TIMER
0277 C8                  INY                 AND DO IT AGAIN
0278 D0 F0               BNE      LBLL       UNTIL 3/4 SEC. BURNS
027A 20 07 E9            JSR      RCHEK      DROP OUT ON ESCAPE
027D 4C 21 02            JMP      LBLD       AND GO TO NEXT GEN.
```

***Table 1:* Locations of external subroutine calls and entry points.**

Location of call	Subroutine description	
D20B	INALL	Input an ASCII character into A
0257	CRLF	Output a carriage return and a line feed
0260	OUTALL	Output ASCII character in A to display
0274	TIMER	Count a delay in microseconds. The delay time is stored in hexadecimal in locations A808 and A809 hex. These locations must be reloaded everytime DE2 is called.
027A	RCHEK	Scans the keyboard, returns to Monitor on ESC, caller on no entry, wait on (space) until another (space) is entered.

***Table 2:* Author's names for various patterns.**

Pattern	Name	Period
__**__	Blinker	2
__**__*	Railroad crossing	2
__**____*	Long RR crossing	2
*__***	Glider	1
*__******	Spaceship	3
__****	Heavy spaceship	3
**********	Blinker seed	—

An asterisk (*) denotes a live cell.

A space (__) denotes a dead cell.

I am using more or less conventional Life terms to denote these patterns. Dr. Millen named the glider in his article.

```
GLIDER EAT GLIDER

12345678901234567890

*  ***     ******  *
 *  ***   **  **  *
  *  ****  *  *  **
   *     ******  *
        **  **   *
       *  *  *  **
        ******  *
       **  **   *
      *  *  *  **
       ******  *
```

Figure 1

```
THE INFINITE BLINKER

12345678901234567890

   **  **  **  **
  *  **  **  **  *
   **  **  **  **
  *  **  **  **  *
   **  **  **  **
  *  **  **  **  *
   **  **  **  **
```

Figure 2

```
FLIP-FLOPS

12345678901234567890

 * **       **  *
  ** *     *  **
 * **       **  *
  ** *     *  **
 * **       **  *
  ** *     *  **
 * **       **  *
```

Figure 3

```
PERIOD 6 OSCILLATOR

12345678901234567890

       *****
      ** * **
     * ***** *
      *  *  *
       ** **
      * * * *
       *****
      ** * **
```

Figure 4

For those unfamiliar with One-Dimensional Life, the rules are as follows:

Rule 1: Birth

Cells that are off but have either two or three neighbors on, go on.

Rule 2: Survival

Cells that are on and have two or four neighbors on, stay on. Those with zero or one neighbors on, die from loneliness; those with three neighbors on, die from overcrowding. What keeps a cell with four neighbors on from dying is not clear. Maybe there is just not enough room to lie down [sic].

It is easy to see that there are no still lives in One-Dimensional Life, and while not ruling out their existence in larger universes, I seriously doubt that glider guns or related "infinite" patterns can be constructed in 20 spaces.

Implementation on Other Systems

Any 6502-based system with 1K or more programmable memory should be able to run this version of One-Dimensional Life with very minor modifications. Table 1 shows the addresses of the jumps to external subroutines used for I/O and the 3/4 second delay between generations. Users with video terminals can run a larger universe by changing the number of locations each array needs, limited only by available zero page space. (Remember to change the index values.) I would estimate that an n-length One-Dimensional Life could support as relatively complex patterns as an $n \times n$ array of a conventional Life universe, so this version can be just as interesting in a far shorter program.

A logical variant would be an 80 cell universe, say, with a moveable window to inspect parts of the pattern. I have not had the time to write such a program, since most of my time must be spent in studies and other activities. I will try to answer any questions or comments pertaining to patterns or program bugs. If anyone has a better scheme for determining generations, I would like to know. Remember, exploration is wide open in this field and anyone could discover that glider gun.

References

1. Jonathan K. Millen Ph.D., "One-Dimensional Life," *BYTE* November 1978, pgs. 68-74.

MICRO

MICRO

New Publications

Mike Rowe
New Publications
P.O. Box 6502
Chelmsford, MA 01824

This column lists new publications received for review and also reports on pertinent publication announcements received from book and periodical publishers. Some works mentioned here may be reviewed by MICRO at a later date.

General 6502

Microcomputer Systems Principles Featuring the 6502/KIM by R.C. Camp, T.A. Smay, and C.J. Triska. Matrix Publishers, Inc. (30 NW 23rd Place, Portland, Oregon 97210), 1978, viii, 548 pages, illustrated, 6 × 9, paperbound.
ISBN: 0-916460-27-4 $15.95

A computer engineering textbook introducing microprocessors and emphasizing hands-on experience with the KIM-1 microcomputer.

CONTENTS: *Preface. Introduction to Microcomputer-based Design*—Evolution of the Microcomputer/Microprocessor Applications; Engineering Design of Microcomputer-based Products; Educational Demands Created by the Microprocessor Objectives of this Book. *General Aspects of Microprocessor-based Systems*—Microprocessors and Microcomputers; Classification of Computers and Computer Systems; General Features of Microcomputer-based Systems; Information Flow in Microcomputers; Central Processor Hardware Elements; Addressing Modes; Microprocessor Instructions Sets; Microprocessor Word Length; Symbolism in Digital Computers; Arithmetic Operations in Microcomputers; Interrupts and Subroutines; Technological Factors in Microprocessors. *The MCS6502 Microprocessor and Peripheral Parts*—Introduction to MCS6502; Programming Model; Data Paths; Concept of Operation of MCS6502 Instructions; Complete Description of Operation Codes; MCS6502 Specifications; Peripheral Interface Chips; Example Problems. *Software Aids*—Introduction; The Software Design Process; Elements of Program Translation; Text Editors; Simulators; Special Program Debug Features; In-circuit Emulation; Logic State Analyzers; Prom Programmers. *Microcomputer Interfacing and System Design*—Introduction; Guidelines for System Design; Miscellaneous Advice on System Design; Interfacing Examples; Input/Output—TTL, Speed, Bits, Serial/Parallel Conversions; Address Maps and Organization—Memory and I/O Selection; System Design Examples. *Introduction to the M6800 Microprocessor*—Introduction; Principal Characteristics; Some MCS6502 and M6800 differences; M6800 Programming; Electrical Characteristics of the M6800; M6800 Microcomputer Example; Example Problems. *Introduction to the I8080 Microprocessor*—Characteristics; I8080 Architecture and Programming Model; Data Paths; I8080 Instruction Set; I8080 Example Program; Electrical Characteristics of the I8080. *An MCS6502-Based Microcomputer—The KIM-1*—Inroduction; What is a KIM-1; The KIM-1 System: (A micro versus a mini); An Example Program; KIM-1 Memory Map and Table; Machine Code Example; Entering Example Code into KIM-1; Execution of Example from the Keyboard; Decimal or Binary Code; KIM-1 Keyboard Key Functions; Operating the KIM-1 via Teletype; Adding an Audio-tape Recorder to the KIM-1; The KIM-1 Display. *Appendices: A.* User's Guide to the MDT 650; *B.* Operating Principles of the KIM-1 Monitor and On-Board I/O hardware.

The Best of MICRO, Volume 3 by *MICRO: The 6502 Journal*. The Best of MICRO Series (ISSN: 0271-8189), Micro Ink, Inc. (P.O. Box 6502, Chelmsford, Massachusetts 01824, 1980, 320 pages, illustrated, 8 3/8 × 10 7/8, paperbound.
ISBN: 0-938222-03-1 $10.00

A selection by MICRO's editors of articles appearing in the magazine's volume 3 (June 1979-May 1980).

CONTENTS: *AIM/SYM/KIM*—(26 articles). *Apple*—Programmers' Aids (6 articles); Graphics Programs (4 articles); Useful Utilities (5 articles); Fun, Games, and Projects (5 articles); Reference and Educational (7 articles); *Ohio Scientific*—(14 articles). *PET/CBM*—(15 articles). *General*—(9 articles). *Author Index.*

General Microcomputer

Funding Report for Microcomputers by Bell & Howell Audio-Visual Products Division (7100 N. McCormick Road, Chicago, Illinois 60645), issued 1980 (undated), 44 pages, 8½ × 11, paperbound.
$5.00

A booklet designed to help U.S. educational institutions identify sources of public and private funding for the acquisition of microcomputer technology applied to *instruction*. The information provided covers conditions as they existed in the fall of 1979.

CONTENTS: Introduction. The Appropriate Federal Titles for Microcomputer Funding. Comments from State Departments of Education. Local Funding and Local Budgets. Proposal Development. Successful Proposals. Recommendations. Federal and State Contacts for Additional Information. Publications. Definitions. Other Sources of Funds.

Son of Cheap Video by Don Lancaster. Howard W. Sams & Co., Inc. (4300 West 62nd St., Indianapolis, Indiana 46268), 1980, 224 pages, paperbound.
ISBN: 0-672-21723-6 $8.95

A sequel to the author's *Cheap Video Cookbook, Son of Cheap Video* (and its predecessor) shows low cost ways of getting alphanumeric and graphics video out of a microcomputer and onto an ordinary television set.

CONTENTS: *Scrungy Video*—How Video Works; A Bottom Line Scrungy Video System. *The Snuffler—Super Simple Transparency*—The Method; Building the Snuffler; A Snuffler Demonstration; Alternate-Field Snuffling; The Best of Both Worlds; Some Perspective. *Custom Characters*—EPROMs as Character Generators; Graphics Chunks; Using EPROMs; Designing a Character Set; Building EPROM Adaptor Module "E"; Checkout. *A Music Display*—The Display Plan; A Character Set; Music Software; Test and Debug; Polyphony. *8080 Cheap Video—Heath H8 Hardware*—Hardware; Speed Doubling Via A9 Switching; Front-panel Interaction; A Keyboard Serial Adaptor. *8080 Cheap Video—Heath H8 Software*—Test Software; Self-Modifying Versus Brute-Force Scans; 1 x 56 Scan Programs; TV Retrace Hassles; More Characters; 12 lines of 80 Characters; 8080 Cursor Software. *Lower-Case Hardware For Your Apple II*—Some Details; Hardware Changes; Initial Checkout. *Lower-Case Software For Your Apple II*—Direct Entry; Four Utility Sequences; A Lower-Case Tester; A Useful Display Program; A Full-Performance Lower-Case Editor; A Full Dual-Case Editing System; Further Hardware Mods. *APPENDIX A:* More Character Generator Details; *APPENDIX B:* Pinouts of selected IC's; *APPENDIX C:* Printed Circuit Patterns.

General Computer

Computer Dictionary by Charles J. Sippl and Roger J. Sippl. Howard W. Sams & Co., Inc. (4300 West 62nd St., Indianapolis, Indiana 46268), 3rd edition 1980, 5 3/8 × 8½, paperbound.
ISBN: 0-672-21652-3 $12.95

The authors call this a "browsing" dictionary, with long definitions and explanations designed to teach users about products, procedures, problems, and applications.

A BRILLIANT FUTURE FOR YOUR AIM-65 WITH THE BANKER MEMORY ™

Your 36K of free address space is the AIM's most valuable and limited resource. With today's large capacity RAM boards, ROM boards, disk systems, video boards, and other expansion accessories it is easy to deplete this resource before the application requirement is satisfied. MTU has solved this problem.

THE BANKER MEMORY contains 32K of RAM, 4 PROM sockets for 2716/2732/2332, a PROM programmer, 40 bits of parallel I/O, and 4 timers from two 6522 I/O chips. Addressing is extremely flexible with the RAM independently addressable in 4K blocks, PROM's independently addressable, and I/O addressable anywhere on a 64 byte boundary (even in AIM's I/O area at AXXX by adding a single jumper to the AIM).

This may sound familiar, but read on! Unlike other AIM compatible memory boards, THE BANKER MEMORY has on-board bank-switching logic! The four 8K blocks of RAM plus the 4 PROM sockets make up 8 **resources**, each associated with a bit in an Enable Register. Through this Enable Register resources may be turned on and off under software control. When a resource is off, its address space is freed for other uses. You can even put BANKER resources at the same address and switch among them for virtually unlimited RAM and PROM expansion! You can even have multiple page zero's and stacks! Do you need 160K byte of memory? It only takes 5 of THE BANKER MEMORY boards and you end up with 5 page zeros and stacks to boot!

There's more! The BANKER MEMORY also incorporates 18 bit addressing which allows for the 256K address spaces of the future. RAM, PROM, and I/O each has its own full 18 bit address decoder which allows these resources to be in different 64K banks. This board and other MTU products, such as our 320 by 200 dot VISIBLE MEMORY and Floppy Disk Controller with 16K DMA RAM, can turn your AIM into a truly powerful 6502 computer that far surpasses the packaged systems in functional performance.

INTRODUCTORY SPECIAL K-1032-1 32K BANKER MEMORY FULLY ASSEMBLED AND TESTED $395.00 ($450.00 as of March 1, 1980) or the K-1032-2 16K RAM only with bank switching and 18 bit address bus only $295.00

Isn't it time you took a closer look at MTU — we offer you power now with an eye to the future.

WRITE OR CALL TODAY FOR OUR 48 PAGE FALL 1980 6502 CATALOG
International requests include $1.00

VISA and MASTERCARD accepted

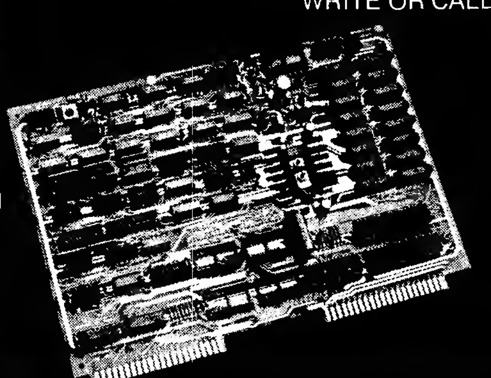


Photo credit:
SUPERNOVA CRAB NEBULA:
Palomar Observatory,
California Institute of Technology

MTU

Micro Technology Unlimited
2806 Hillsborough Street
P.O. Box 12106
Raleigh, NC 27605, U.S.A.
(919) 833-1458

Increase KIM-1 Versatility at Low Cost

If KIM's primary address decoder is moved off-board to a small expansion board, it becomes possible to add other I/O devices in page 5 without developing bus contention with KIM's regular I/O ports. Also, further expansion is made easier, and KIM's whole memory map is more efficiently used.

Ralph Tenny
P.O. Box 545
Richardson, Texas 75080

I/O PORT	START ADDR.	END ADDR.	ADDR. LINES
0	1400	147F	K0/8, A0-A7
1	1480	14FF	K1/8, A0-A7
2	1500	157F	K2/8, A0-A7
3	1580	15FF	K3/8, A0-A7
4	1600	167F	K4/8, A0-A7
5	1680	16FF	K5/8, A0-A7

***Table 1:* The new decoder chip allows addition of six more I/O ports, each 128 bytes wide. Any 6502 family programmable peripheral device can be used, as well as 128 × RAM or 128 × PROM.**

The KIM-1 memory map is fairly well organized for those who need a 5K+ system, provided no additional I/O channels are needed (see figure 1). However, the area presently dedicated to I/O (1400-177F) is not exclusively decoded. That is, devices enabled by K5 will encounter bus contention from the 6530 I/O registers without further decoding. At first, it is almost obvious that any additional decoding must result in trace cutting on the KIM board, as a minimum.

If U4 (74LS145) is moved off-board and addressed by the same lines as before, it is possible to add another 74LS145 or 7445 (same pinout, same truth table) driven by A7, A8 and A9 and enabled by K5 (see figure 2). The eight outputs (shown as K0/8 through K7/8) each decode 128 bytes of the 1K bytes enabled by the previous connection of K5. The new K5 is made up of K6/8 and K7/8 connected together and connected to KIM via pin H of the Application connector. The pull-up resistor which served the original K5 now serves the new K5.

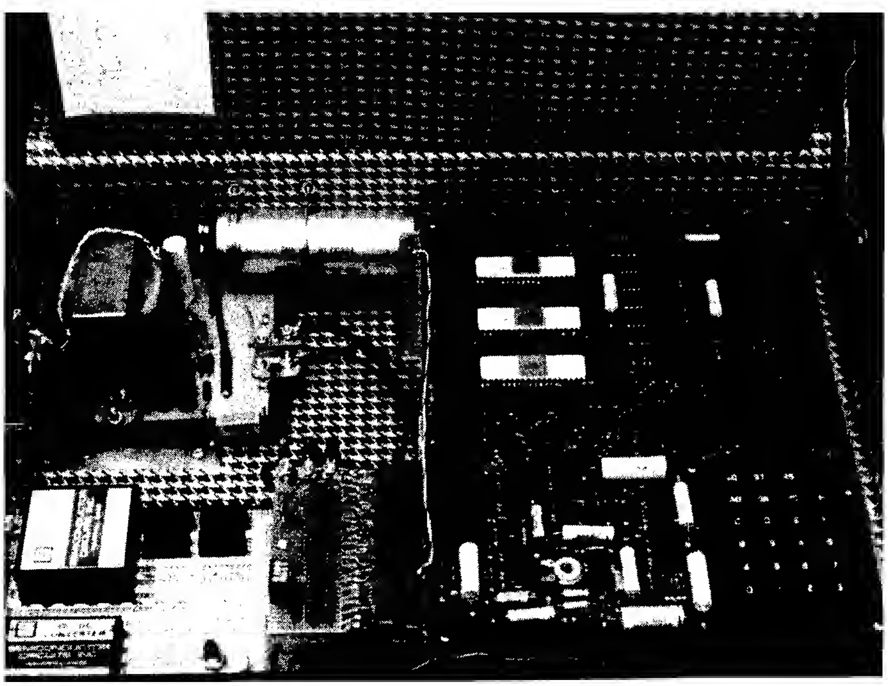
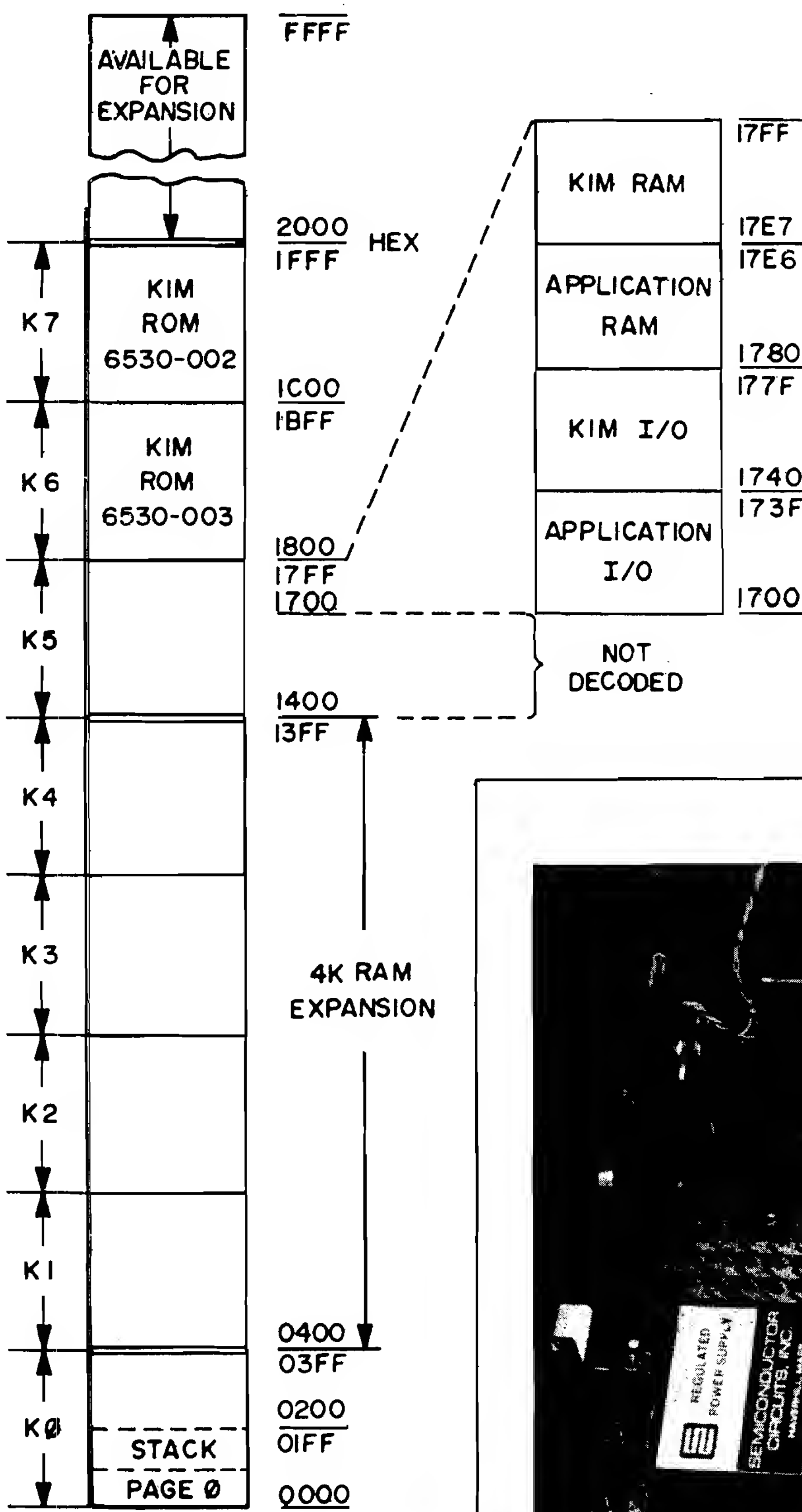
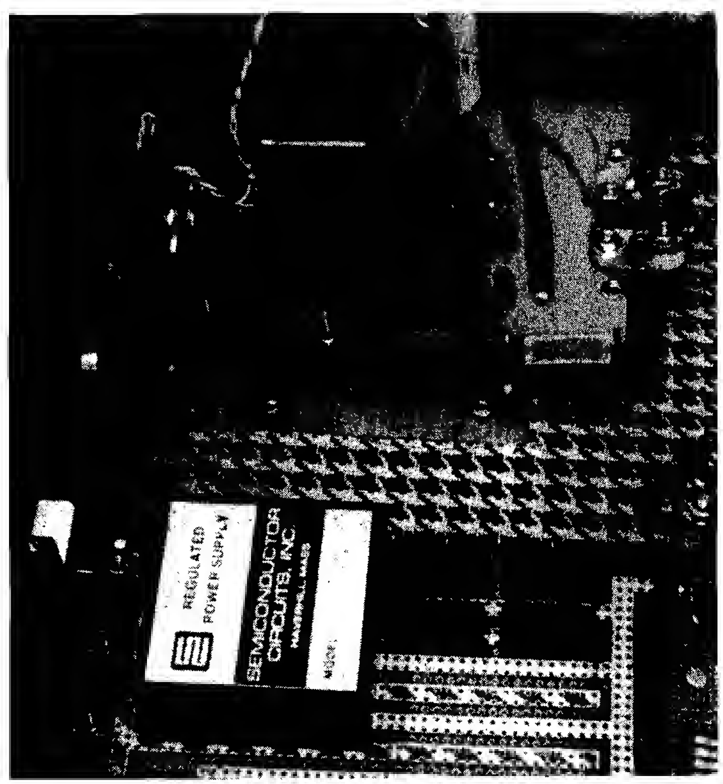


Figure 1: **KIM-1 Memory Map; note that the memory area decoded by K5 is not totally decoded so that any other ports in the same area will conflict with existing KIM ports.**





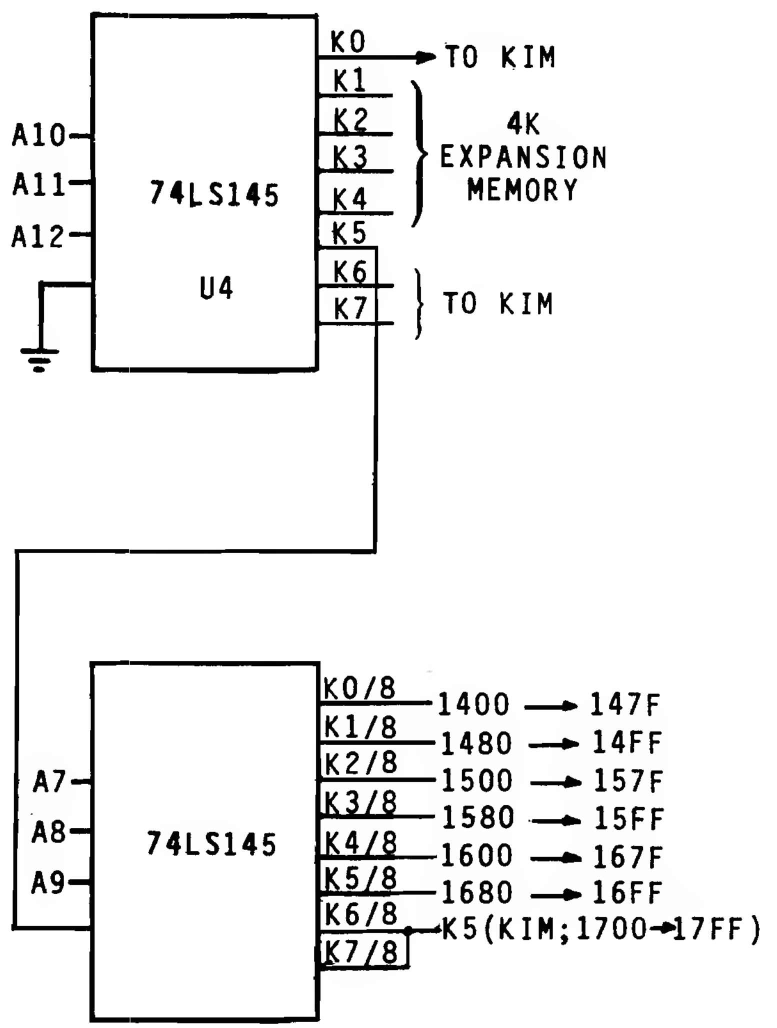
Since K1, K2, K3 and K4 are needed off-board for memory expansion, they do not need to be returned to KIM. K0 and K7 (from the re-located U4) are connected to their original exit point from KIM (pins B and J, respectively, on the Application connector). Since K6 originally did not exit on the Application connector, use K4's old home (pin F). At the old U4 socket, jumper pin 5 to pin 7 so that K6 is routed over its old path.

At this point, KIM's memory map has not really been changed. However, there are now six 128 byte blocks which are decoded for whatever may be needed. If you need more I/O, it is very easy to add up to 96 additional I/O lines by using six 6522's, 6520's, etc.

Table 1 summarizes the connections for these devices, along with the addresses they will respond to. Note that so many additional devices will probably cause overload on the processor address lines, so address buffers on the lower seven lines would be needed. If all 16 address lines are buffered at the expansion connector, there will be adequate drive for almost any desired expansion.

If additional memory expansion is needed, simply follow the guidelines in the *KIM-1 User Manual*, Chapter 6. That scheme is completely compatible with the circuitry shown here except that U4 has been moved off-board and the total drive load on the processor is less than before.

***Figure 2:* With U4 moved off-board so that K5 can enable a second decoder, it is possible to create six new I/O ports, each with 128 bytes of address space.**



MICRO

Microbes and Updates

Mike Rowe
Microbes & Updates
P.O. Box 6502
Chelmsford, MA 01824

In our listing of Roger C. Crites' "Stuffit" (MICRO 31:45), we inadvertently omitted line 340 and have a correction to line 90:

```
90   GET A$:1F A$ = " " GOTO 90
340  ?"STRIKE ANY KEY TO
     CONTINUE"
```

Lines 415 and 465 (not 15 and 65) are the ones that are ineffective on new PETs.

Also, the following abbreviations were used in the BASIC listing:

?	PRINT
cs	clear screen
ch	cursor home
cl, cr, cd, cu	cursor left, right, down and up
[10 cd]	10 successive cursor downs
rv	reverse field
of	off

John P. Hill of Greenville, South Carolina, discovered that we omitted two lines from listings in Brooke W. Boering's "Multiplying on the 6502" (31:71). In figure 2, the portion starting at RMUL4 should read:

```
1032 66 53 RMUL4 ROR XTNDH
1034 66 52       ROR XTNDL
1036 66 51       ROR ACH
1038 66 50       ROR ACL
103A 88          DEY
103B D0 E3       BNE RMUL2
103D 60
```

In figure 3, the portion beginning at MUL3X should read:

```
104D A5 50 MUL3X LDA ACL
104F 4A          LSR A
1050 90 0E       BCC MUL4X
```

The addresses are all incremented by two and the next to last line now should read:

```
1068 D0 E3       BNE MUL3X
```

Eugene Weiner (Weiner, DeJong and Lenth), "A Random-Character Morse Code Teacher for the Aim 65 (31:21), has pointed out two errors in the listings that were not in the original:

```
0F77 B9 00 0F LDA $0F00,Y
instead of
0F77 AD 00 0F LDA $0F00
```

In the BASIC listing, line 151 should read: Z = S:X = 0, not Z = S:X + 0.

Apple* Software

From Instant Software

Santa Paravia and Fiumaccio

Buon giorno, signore!

Welcome to the province of Santa Paravia. As your steward, I hope you will enjoy your reign here. I feel sure that you will find it, shall we say, profitable.

Perhaps I should acquaint you with our little domain. It is not a wealthy area, signore, but riches and glory are possible for one who is aware of political realities. These realities include your serfs. They constantly request more food from your grain reserves, grain that could be sold instead for gold florins. And should your justice become a trifle harsh, they will flee to other lands.

Yet another concern is the weather. If it is good, so is the harvest. But the rats may eat much of our surplus and we have had years of drought when famine threatened our population.

Certainly, the administration of a growing city-state will require tax revenues. And where better to gather such funds than the local marketplaces and mills? You may find it necessary to increase custom duties or tax the incomes of the merchants and nobles. Whatever you do, there will be far-reaching consequences. . . and, perhaps, an elevation of your noble title.

Your standing will surely be enhanced by building a new palace or a magnificent *cattedrale*. You will do well to increase your landholdings, if you also equip a few units of soldiers. There is, alas, no small need for soldiery here, for the unscrupulous Baron Peppone may invade you at any time.

To measure your progress, the official cartographer will draw you a *mappa*. From it, you can see how much land you hold. how much of it is under the plow and how adequate your defenses are. We are unique in that here, the map IS the territory.

I trust that I have been of help, signore. I look forward to the day when I may address you as His Royal Highness, King of Santa Paravia. *Buona fortuna* or, as you say, "Good luck". For the Apple 48K.

Order No. 0174A $9.95 (cassette version).
Order No. 0229AD $19.95 (disk version).

TO ORDER

SEE YOUR LOCAL INSTANT SOFTWARE DEALER OR USE THE ORDER FORM BELOW

For Fast Service *call now* Toll-Free 1-800-258-5473

Apple Cassettes

0018A	Golf	$7.95
0025A	Mimic	$7.95
0040A	Bowling/Trilogy	$7.95
0073A	Math Tutor I	$7.95
0079A	Oil Tycoon	$9.95
0080A	Sahara Warriors	$7.95
0088A	Accounting Assistant	$7.95
0094A	Mortgage w/Prepayment Option/ Financier	$7.95
0096A	Space Wars	$7.95
0098A	Math Tutor II	$7.95
0174A	Santa Paravia and Fiumaccio	$9.95
0148A	Air Flight Simulation	$9.95

We Guarantee It!

Instant Software Guarantee

OUR PROGRAMS ARE GUARANTEED TO BE QUALITY PRODUCTS. IF NOT COMPLETELY SATISFIED YOU MAY RETURN THE PROGRAM WITHIN 60 DAYS. A CREDIT OR REPLACEMENT WILL BE WILLINGLY GIVEN FOR ANY REASON.

109

Name ____________________

Address ____________________

City ____________ State ________ Zip ________

☐ Check ☐ Money Order ☐ VISA ☐ AMEX ☐ Master Charge

Card No. ____________ Exp. Date ________

Signed ____________ Date ________

Order your Instant Software today!

Quantity	Order No.	Program name	Unit cost	Total cost
		Shipping and handling		$1.00
			Total order	

Instant Software™ Inc.

Peterborough, N.H. 03458

MICRO Classified

Programmer Fatigue?

SYM—BUG/MONEX adds 15 commands to SYM's repertoire including an interactive trace/debug. Cassette @ $0200 or $3800: $19.95. EPROM (2716-5v) @ $F000-$F7FF: $39.95. Commented source listing: $9.95. RAE-1(/2) FORMAT CASSETTE: $35 (requires 8K). Custom assembly add $2.00. Foreign add $2.00. SASE for more information.

Jeff Holtzman
6820 Delmar-203
St. Louis, Missouri 63130

PET Machine Language Guide

Comprehensive manual to aid machine language programmer. More than 30 routines are fully detailed so that the reader can put them to immediate use. OLD or NEW ROMS. $6.95 + .75 postage. VISA & Mastercharge accepted.

Abacus Software
P.O. Box 7211
Grand Rapids, Michigan 49510

OHIO SCIENTIFIC

Animated Moon Lander game for C1 and C4. View a series of high resolution looking lunar vistas as you descend from 120 miles. Accurate instrument readings. A cartoon landing sequence rewards your successful landing. See what OS1 graphics can do! Other programs available. $9.95 cassette 8K; $12.45 disk 24K.

Earthship
P.O. Box 489
Sussex, New Jersey 07461

OHIO SCIENTIFIC

Catchword, a multi-player, competitive word game uses the computer to generate letters and point values. Letters are up for grabs by any player. The idea is to make as many words as possible in crossword form on your board. The computer recognizes and penalizes you for all sorts of sloppy playing. C1 and C4. $9.95 cassette 8K; $12.45 disk 24K. Other programs available.

Earthship
P.O. Box 489
Sussex, New Jersey 07461

AIM-65 High Quality Power Supply

Designed to Rockwell's specifications. Overvoltage protection, fuse, switch, pilot light, line cord, cable — all included. Handsome all metal case. Satisfaction or return unit within 10 days for full refund. VISA/MC. Check (2 weeks to clear). $64.95 plus shipping (5 lbs).

CompuTech,
Box 20054
Riverside, California 92516

Spanish Hangman

2,000 SPANISH words and sentences taught in a fun way on the Apple. Send for your school's free 30-day evaluation diskette, from:

George Earl
1302 South General McMullen
San Antonio, Texas 78237

APPLE Banner Printer Program

Prints banners with any message up to 70 characters with large letters, which are 70 columns high. Machine language. DOS 3.2 disk or cassette $8.00, or send .30 in stamps for sample printout and more information.

Philip Bryan
529 West Street
Park City, Illinois 60085

Turnkey Medical Billing System

Interactive data entry. Automated record management. Outputs: Patient statements, Universal Claim Forms, financial reports. Customized by user-developed text files. Requires Apple, Applesoft, printer. One disk drive manages 200 accounts; 2 drives — 500 accounts. $350 for programs and 17+ pp. documentation.

Jerome B. Blumenthal, M.D.
7500 E. Hellman
Rosemead, California 91770

Atari · Pet Owners

KINETIC DESIGNS has software for you! Games and Simulations, Music, Astronomy, Ham Radio, home use, utility, and many others. MTU Visible Memory and music programs now available! Prices start at only $2.50! (Atari $3.50) Guaranteed! Send SASE for full details.

Kinetic Designs
401 Monument Road #171
Jacksonville, Florida 32211

AIM/KIM/SYM

NBS Computing gives you time! A battery backed-up clock-calendar board that runs on the application bus. The clock will run for months without power and can generate interrupts on SYM systems. $69.95 assembled, $34.95 bare board. Both include drivers.

NBS Computing
1674 E. M-36
Pinckney, Michigan 48169

Save Money — You Can!

By building your own computer interfaces. 80% savings. Send $3.95 for simple how-to package today.

ADS
Box 9770
Jacksonville, Florida 32208

PET String Flip

This routine solves the problem of upper and lower case inversion when using CBM 2022 and 2023 printers with OLD ROM PETs. The method is to invert the characters in the string area of RAM.

James Strasma
120 West King Street
Decatur, Illinois 62521

PET owners with old ROMs (2.0) and a Commodore CBM 2022 or 2023 printer have a problem. The printers do a fine job with lowercase and uppercase printing. However, what appears on the screen as uppercase comes out on the printer in lowercase, and vice versa. This is due to the non-standard ASCII codes used by the old ROM PET. Some new ROM PET owners may have the same problem, if they use a printer interface that was designed to correct the output of an old ROM PET.

According to Commodore's *PET Users Club Newsletter* (issue #9), the recommended solution for this problem is to order upgrade ROMs (3.0). This is a good solution. It gives the user a monitor in ROM, and access to the non-maskable interrupt (for a warm start after a "crash"). However, it costs close to a hundred dollars to upgrade. Also, many PET owners are fiercely loyal to their old ROMs, and would rather fight than switch. For those owners, and for those with new ROMs and a printer interface designed for old ROMs, STRING FLIP offers another solution.

```
                STRING FLIP
                OLD ROM PET'S WITH CBM PRINTERS

036C            BTMSTR  *      $0082    BOTTOM OF STRING SPACE
                ($0030 WITH 3.0 ROM'S)
036C            HIMEM   *      $0086    TOP OF RAM MEMORY
                ($0034 WITH 3.0 ROM'S)
036C            POINTR  *      $0001    INDIRECT POINTER

033A                    ORG    $033A    SECOND CASSETTE BUFFER

033A A0 00              LDY    #$00     CLEAR INDEX
033C A5 82              LDA    BTMSTR   COPY BOTTOM OF
033E 85 01              STA    POINTR   STRINGS ADDRESS
0340 A5 83              LDA    BTMSTR   +01 INTO POINTER
0342 85 02              STA    POINTR   +01 LOCATION.

0344 A5 02      DONE    LDA    POINTR   +01 CHECK WHETHER
0346 C5 87              CMP    HIMEM    +01 THE POINTER HAS
0348 30 07              BMI    NOTDON   REACHED THE
034A A5 01              LDA    POINTR   TOP OF RAM
034C C5 86              CMP    HIMEM    MEMORY YET.
034E D0 01              BNE    NOTDON   IF SO,
0350 60                 RTS             RETURN TO BASIC.

0351 B1 01      NOTDON  LDA    (POINTR),Y LOOK AT THE
0353 AA                 TAX             NEXT STRING CHARACTER.
0354 29 7F              AND    #$7F     IGNORE ITS CASE FOR NOW.
0356 C9 41              CMP    #$41     IS IT A LETTER?
0358 90 09              BCC    UPPNTR
035A C9 5B              CMP    #$5B     IF NOT,
035C B0 05              BCS    UPPNTR   DON'T CHANGE IT.
035E 8A                 TXA             REMEMBER THE CASE.
035F 49 80              EOR    #$80     FLIP CASE, & PUT BYTE
0361 91 01              STA    (POINTR),Y BACK IN STRING AREA.

0363 E6 01      UPPNTR  INC    POINTR   UP THE POINTER
0365 D0 DD              BNE    DONE     & LOOP BACK.
0367 E6 02              INC    POINTR   +01
0369 D0 D9              BNE    DONE
036B 00                 BRK             CAN'T GET HERE!
```

This is a short program, just 50 bytes long. It is entirely relocatable. It inverts the bit of each byte that indicates uppercase or lowercase. It does this throughout the part of memory known to hold string data. However, it only changes those values within the range of the alphabet. This allows the user to invert most data before sending it to the printer. After printing, the routine may be called again to flip the data back to the normal screen form.

I use STRING FLIP this way: load STRING FLIP, alone or within a program as data; define a variable, flip, to remember we've flipped; before each printer routine, have a line such as

```
300IF FLIP < 1 THEN SYS(826):
     FLIP=1:REM  INVERT
     CASE
```

Then on return to screen mode, have a similar line:

```
100IF FLIP > 0 THEN SYS(826):
     FLIP=0:REM  NORMAL
     CASE
```

This routine should correct most of your printouts. If you find a string that still prints incorrectly, most likely it is defined within the BASIC program, rather than in the string area. You can correct this by redefining it, as in this example:

```
200a$="Sorry, Try Again"
   :a$=a$+" "
```

Enjoy STRING FLIP in all your word-processing and data-handling programs.

Jim Strasma is an associate pastor at a large United Methodist Church in central Illinois. He developed an interest in personal computers when he accidentally wandered into one of the very first computer stores in New York City, in January 1977. Currently, he is developing church-related software. He is also organizing a users group for persons with any of the assemblers by Carl Moser. MICRO readers interested in either effort are welcome to contact him.

MICRO™

MICRO PET Vet

By Loren Wright

First, I would like to revive the "Panic Button" routine by Michael Riley (*PET User Notes*, Vol. I, #7). On the OLD PET, it allows you to recover from a crash without losing memory. Since it is a machine language routine that has to be loaded in, it is certainly less convenient than pressing a reset button. However, we all know that PETs don't have reset buttons. 2.0 ROM PETs can't even support them. Operation is simple. Load the program in, and initialize with SYS 826. Now run your suspect program—BASIC or machine language. If it crashes, just press the RUN/STOP and RVS/OFF keys together and you're alive again. Before running again, the Panic feature must be reinitialized with SYS 826. While initialized, normal cassette operation is not possible, so be sure to deactivate the "Panic Button" by pressing the two keys, before attempting any cassette maneuvers.

Briefly, the way it works is to divert the program flow to a routine that checks for the two keys. Even when the PET is crashed, it must pass through the hardware interrupt vector at \$0219,\$021A. When the keys are pressed, the vector is restored to its original condition. You probably wouldn't want to load this in every time you want to play a game, but it could save a lot of frustration during program development.

```
033A 78         INIT     SEI
033B A9 47               LDA  #$47
033D 8D 19 02            STA  $0219
0340 A9 03               LDA  #$03
0342 8D 1A 02            STA  $021A
0345 58                  CLI
0346 60                  RTS
0347 A9 F9      PANIC    LDA  #$F9
0349 8D 10 E8            STA  $E810
034C AD 12 E8            LDA  $E812
034F C9 EE               CMP  #$EE
0351 F0 03               BEQ  RSTORE
0353 4C 85 E6            JMP  $E685
0356 A9 85      RSTORE   LDA  #$85
0358 8D 19 02            STA  $0219
035B A9 E6               LDA  #$E6
035D 8D 1A 02            STA  $021A
0360 6C 1B 02            JMP  ($021B)
```

3.0 and 4.0 ROM PETs can support a two-button reset device that will restore control without losing RAM contents. To implement a warm reset, pin 5 of the parallel user port must first be grounded. Then the reset line (Memory Expansion J4—pin 22) is momentarily grounded, and pin 5 of the user port can now be released. This will put you in the monitor. You must first exit the monitor, and then re-enter it, to restore normal operation. One commercial unit available is the Uncrasher from International Technical Systems (Woodbridge, Virginia, \$14.95). Also, the new 1st MATE memory expansion board from The Computerist (Chelmsford, Massachusetts) supports this feature with switches and debouncing circuitry. Others are sold by Gord Reithmeyer in Canada and Qwerty Computer Services in Great Britain.

Avoid Accidental INPUT Exit

This is one easy way to prevent exiting an INPUT statement with an accidental RETURN. My favorite character to use for this is a shifted '?' (ASCII 191), but any can be used.

```
10 INPUT "crcrcr▚clclcl";X$
20 IFX$="▚"THEN10
```

An escape is still possible by moving the cursor off the INPUT line and hitting return, but this is more difficult to do accidentally.

Commodore's Computer Shows

Commodore's own computer show in Philadelphia (November 13 and 14, 1980) was attended by more than 16,000 people. Commodore literally brought a truckload of computers, so that all of the exhibitors had the equipment they needed. In addition there were many computers available to the public for trying out programs and playing games. Based on the success of this show, Commodore will take the show to Boston, February 7-8, 1981, at the Boston Sheraton. The next location is New York, probably by the end of February.

Publications

In contrast to the U.S., Commodore occupies the dominant position in Europe in the personal computer market. As a result, there are many Commodore-oriented companies producing good software and hardware. *Printout Magazine* is a 48-page publication covering the PET and CBM exclusively. A sample issue can be had for \$3.00 postpaid, or a subscription for \$36.00.

Printout Magazine
P.O. Box 48
Newbury RG16 OBD
Berkshire, Great Britain

The Central Illinois PET Users has established a free publication called *The Midnight Software Gazette*, to fill the need for short, timely reviews of hardware and software products for the PET. Send a self-addressed, stamped envelope (2 oz.—\$.28 U.S. and Canada) for the current issue. When you get it, be sure to make copies and distribute them to your friends!

Central IL PET Users
c/o Jim Strasma
3838 Benton Drive
Decatur, Illinois 62526

To Authors and Would-be Authors

MICRO has still not been overwhelmed by PET manuscripts, so it is time to point this out to any hesitant authors. Articles do not need to be on the most advanced topics. Good treatment of an elementary subject is as good, or better. Articles describing a particular computer application, in business, industry, home, education, and others, are encouraged. Also, remember, there is room on the PET Vet page for your short items, and for me to answer your questions.

One of my continuing objectives is to make all PET programs published in MICRO usable by owners of all three sets of BASIC ROMs. Authors can help by keeping this in mind when submitting manuscripts. In some cases, this means writing the program a little differently, so it *will* run on all machines. In other cases, it will mean providing a separate list of the alternate page zero and PET subroutine addresses.

Programs that can run on more than one manufacturer's computer are especially sought by MICRO, but it is not enough to say your program can be easily modified—*include the modifications!*

Another way authors can help is by submitting tapes or diskettes with manuscripts. This not only aids in testing and evaluation of programs, but in most cases, allows us to produce letter-quality listings directly from our PETs, without adding the time-consuming and error-prone procedure of keying programs in by hand.

A C1P Sound Idea

This hardware addition creates a bell tone for the C1P or Superboard II.

David A. Ell
19926 N.E. Halsey
Portland, Oregon 97230

Ohio Scientific Superboard II and C1P users, does the idea of a bell tone to keep you company while you are programming your C1P or Superboard appeal to you? I have found the bell tone to be quite useful while I am programming. It sounds when I have a line feed or a return on my video, therefore I don't need to look at the screen to see if I have hit my return key. The tone also sounds when I am loading to, or listing from, my cassette, giving me time to relax or do something else, while the machine does its own monitoring. When the tone stops sounding I know the program is loaded or listed, whichever the case may be.

The time it takes to install this bell tone to your C1P or Superboard is very minimal and is quite inexpensive. Here is what you will need:

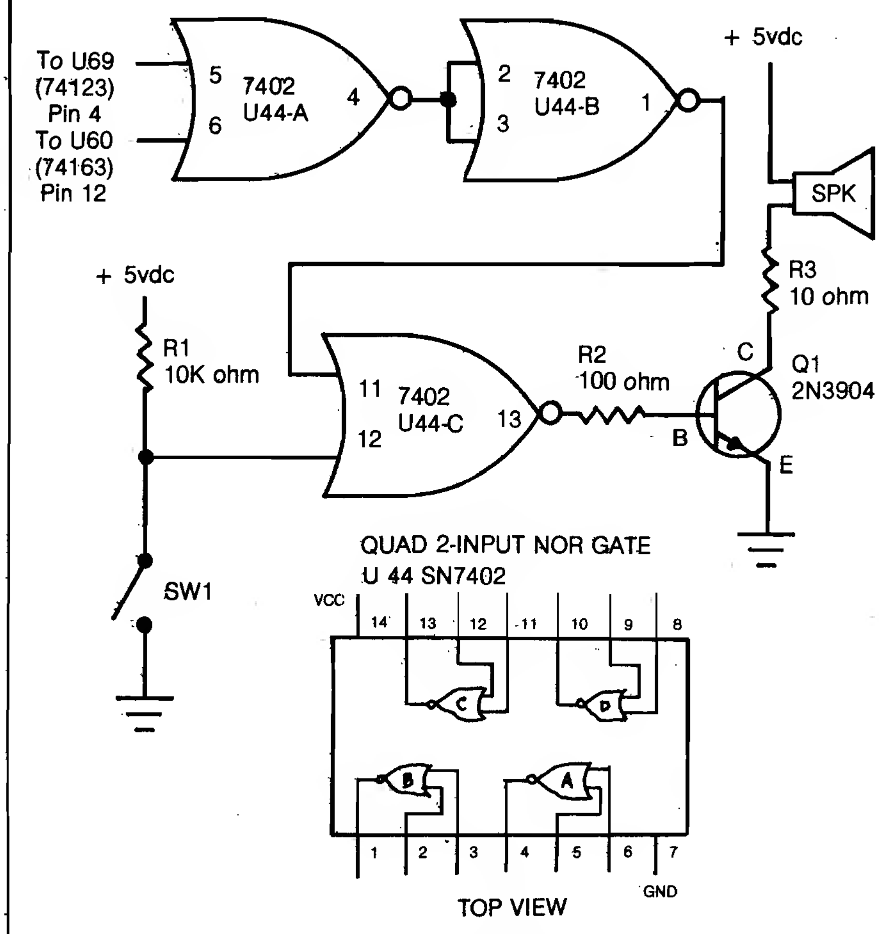
- 1 - 14 pin IC socket (optional)
- 1 - 7402 IC
- 1 - 3904 NPN transistor
- 1 - small speaker
- 1 - 100 ohm resistor
- 1 - 10 ohm resistor
- 1 - 10K resistor
- A soldering iron, solder, and some hook-up wire.

The 7402 IC should, if possible, be a regular TTL gate, since lower powered gates don't have as much drive power. You will need the higher drive power, because gate C of the 7402 is used to drive a 3904 NPN transistor, which is used as an audio output transistor.

One input on gate A of the 7402, a two input NOR gate, is connected to the systems clock at U60, pin 12. The second input of gate A is connected to a one shot in the video system, at U69, pin 4, which holds it high. The high at this input keeps the output low. When you receive a video line feed or return, the one shot goes low momentarily, putting the clock divider frequency from U60, on the output, then going high again ending tone.

Gate B is used as a buffer, to eliminate unwanted noise. Both inputs of gate B are tied together to the output of gate A.

Gate C is used to re-invert the output of gate B and can also be used to switch the audio tone off when it is unwanted. One input of gate C is connected to the output of gate B. The second input is pulled high through a 10K resistor. Pulling this input low, through a switch, turns on the audio tone.



NOTE: Almost any NPN transistor will work.

POWERSOFT
P. O. BOX 157
PITMAN, NEW JERSEY 08071
(609) 589-5500

I found that the easiest way to put the 7402 gate on the C1P is to use the proto position just next to the crystal, which is listed as U44. I used the 14 pin DIP socket, which I placed in the U44 proto position. There are two extra holes, which can be used as connection points for the transistor or for the resistors.

I used wire-wrap wire wherever hook-up wire was needed in the following connections. Pin 5 of the 7402, gate A, connects to pin 4 of U69. Pin 6 connects to the clock chain at U60, pin 12. The output of gate A, pin 4, connects to gate B, pins 2 and 3. The output of gate B, pin 1, connects to pin 11, one input of gate C. The second input of gate C, pin 12 is pulled to five volts, through the 10K resistor. The switch, SW1, is connected between pin 12 and ground. The output of gate C, pin 13, is fed through a 100 ohm resistor to the base of the transistor, to amplify the output for the speaker. R2 is connected from 5 volts to the collector of the transistor and the emitter is connected to ground.

The transistor can be mounted at any point. I put a connection strip on my speaker and connected the transistor on the speaker itself. The circuit is not critical, therefore almost any general purpose NPN transistor will work.

Pin 7 of the 7402 is the power ground, and pin 14 is the +5 volt supply. The remaining pins need not be connected to anything.

If you do not wish to use the on-off switch in the circuit, you can eliminate the pull-up resistor, R1, and pin 12 can be jumpered to pin 11.

After you carefully check your connections for solder bridges, you are ready to run.

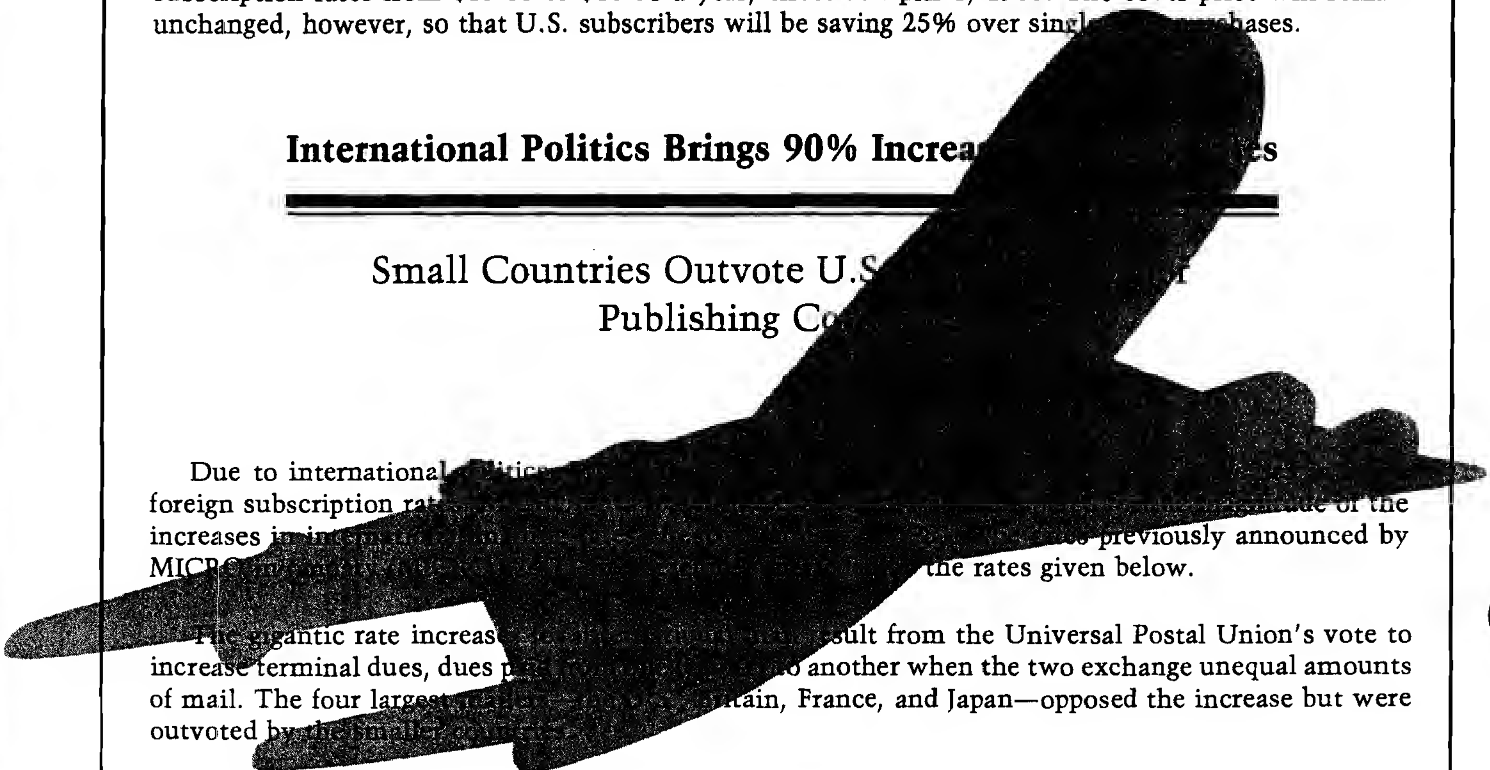
David Ell is a technical serviceman, who recently moved to Portland, where he is currently employed at Western Skyways in the instrument service division. He is a member of the Ohio Scientific Users Group Northwest. He is also involved with amateur radio. Dave's computer is an Ohio Scientific Superboard II with various modifications, working in hand with a Sperry Univac DCT 500 ASR printer terminal system.

Some of the other things Dave has come up with are a 16 line I/O port, reverse video, selectable baud rate, piggy-backed memory, and a number of other usable ideas.

MICRO™

Why Magazine Subscription Rates Go UP, UP, UP!

Due to general inflationary pressures and increased mailing costs, MICRO must increase U.S. subscription rates from $15.00 to $18.00 a year, effective April 1, 1980. The cover price will remain unchanged, however, so that U.S. subscribers will be saving 25% over sing[illegible]hases.



International Politics Brings 90% Increa[illegible]s

Small Countries Outvote U.S[illegible] Publishing C[illegible]

Due to international [illegible] foreign subscription ra[illegible]e of the increases i[illegible]previously announced by MIC[illegible]the rates given below.

The gigantic rate increas[illegible]ult from the Universal Postal Union's vote to increase terminal dues, dues p[illegible] another when the two exchange unequal amounts of mail. The four larg[illegible]ain, France, and Japan—opposed the increase but were outvoted [illegible]

MICRO™ Works to Reduce Rates!

To save European subscribers from the full impact of the international rate increase, MICRO has engaged a European airline which will air freight the magazine to Europe and there post it by surface mail. Air Mail subscribers should receive their copies no later than the middle of each month.

MICRO will attempt to make similar arrangements for other geographic areas. If so, subscriptions made at the new rates will be extended by the amount of the savings achieved.

MICRO's New Annual Subscription Rates

(Effective Immediately)

	Current	New
U.S. (Effective 4/1/81)	$15.00	$18.00
International Surface Mail		
Anywhere outside the U.S.	18.00	21.00
International Air Mail		
Europe	33.00	36.00
Middle East	39.00	42.00
Africa		
North	39.00	42.00
Central	39.00	51.00
South	39.00	60.00
Mexico, Central America	27.00	39.00
South America	33.00	51.00
Far East, Australasia	39.00	60.00

Does Anybody Really Know What Time it is?

Add a real time, non-interruptable hardware clock/calendar to your 6502 system using a new clock chip and you will be as close to knowing as anyone can be.

Randy Sebra
54 Krouse Court
Aberdeen, Maryland 21001

A hardware real-time clock has several advantages over a software real-time clock. First, keeping time does not require interrupt driver software, thereby saving machine time overhead and RAM space. Next, the circuit described here can generate its own interrupts to the microprocessor if regularly spaced interrupts are needed. Finally, and perhaps most significant is that being non-interruptable with its battery backup, the time only has to be set when starting up the first time. Neither turning off the microprocessor system nor power outages affect the keeping of time.

The MSM5832

The MSM5832 from OKI Semiconductor is a CMOS clock/calendar chip made especially for bus-oriented microprocessor applications. Due to its special design, it offers many advantages over other types of conventional clock circuits when used with a microprocessor as a non-interruptable clock/calendar.

The MSM5832 keeps track of seconds, minutes, hours, day of the week, date, month, and year. Data is read and written by using a four bit bi-directional bus, when addressed by a four bit address bus. Table 1 shows the function of each address. Notice that in

ADDRESS INPUTS A0	A1	A2	A3	INTERNAL COUNTER	DATA I/O D0	D1	D2	D3	DATA LIMITS	NOTES
0	0	0	0	S 1	*	*	*	*	0-9	S1 or S10 reset to zero whenever write is executed
1	0	0	0	S 10	*	*	*		0-5	
0	1	0	0	MI 1	*	*	*	*	0-9	
1	1	0	0	MI 10	*	*	*		0-5	
0	0	1	0	H 1	*	*	*	*	0-9	
1	0	1	0	H 10	*	*	†	†	0-1 0-2	D2="1",PM D3="1",24 Hour D2="0",AM D3="0",12 Hour
0	1	1	0	W	*	*	*		0-6	
1	1	1	0	D 1	*	*	*	*	0-9	
0	0	0	1	D 10	*	*	†		0-3	D2="1",29 days in month 2(2) D2="0",29 days in month 2
1	0	0	1	MO 1	*	*	*	*	0-9	
0	1	0	1	MO 10	*				0-1	
1	1	0	1	Y 1	*	*	*	*	0-9	
0	0	1	1	Y 10	*	*	*	*	0-9	

(1) * Data valid as "0" or "1".
Blank does not exist(unrecognized during WRITE and held at "0" during READ).
† Data bits used for AM/PM, 12/24 Hour and leap year.
(2) If D2 previously set to "1", upon completion of month 2 day 29, D2 will be internally reset to "0"

***Table 1:* Functions**

CONDITIONS	OUTPUT	REFERENCE FREQUENCY	PULSE WIDTH
HOLD = L	D0(1)	1024 Hz	duty 50%
READ = H	D1	1 Hz	122.1 us
CS = H	D2	1/60 Hz	122.1 us
A0-A3 = H	D3	1/3600 Hz	122.1 us

(1) 1024 Hz signal at D0 not dependent on HOLD input level.

***Table 2:* Reference Signal Outputs**

addition to being able to program through software either a 12 or 24 hour format, leap years are handled quite easily. Leap years are controlled by bit D2 of address 8. When set, it gives the second month of the year a 29th day, and after the 29th day has elapsed, the bit is automatically cleared. The bit may be set any time after the second month of the previous year, and before the end of the second month of the leap year.

Another feature is a manual ±30 second correction input. Perhaps the most unique and useful feature is the HOLD control which allows read/write operations to occur with the counters being held static, without disturbing the accuracy of the real time. Additionaly, four different interrupt outputs are available to the microprocessor, as shown in table 2. Finally, the chip will operate on a battery back up as low as 2.2V with a power dissipation of less than 90uW, making long term backup quite attractive and economical.

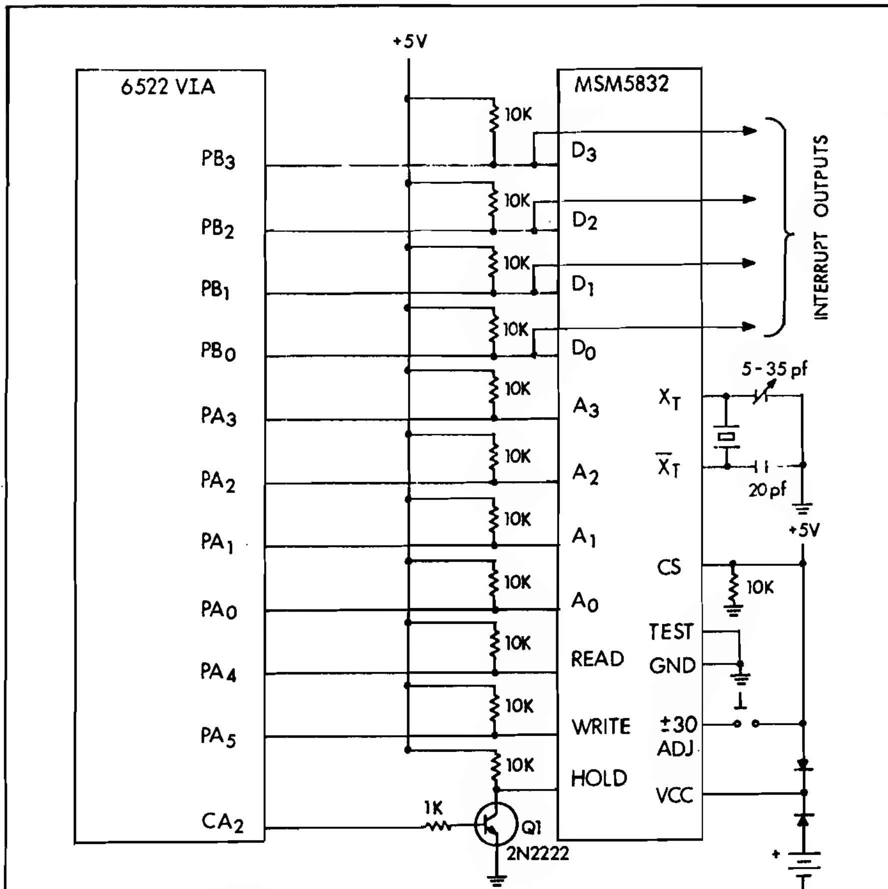


Figure 1: Clock Circuit

Functions

The functions of the clock/calendar are best described on an individual basis as follows.

Oscillator (XT,$\overline{XT}$)

A 32.768KHz(2^{15}) crystal is connected to an internal, stable oscillator to form an accurate time base. The two parallel capacitors, one of which is a trimmer, allow the oscillator to be tuned quite precisely.

A0-A3

These are the address inputs which are used to select the internal counters to be set or read on a read or write operation.

D0-D3

These are data inputs or outputs, depending on whether a read or write operation is being done. They are tri-state bi-directional ports controlled by the READ and WRITE controls.

Chip Select

This determines whether the inputs and outputs are active or inactive. Connecting the CS to Vcc activates the inputs and outputs, while connection to ground disables them. In the circuit in figure 1, the CS is permanently connected to +5V from the microprocessor system for battery backup configuration. When the main system is turned off, this disables all

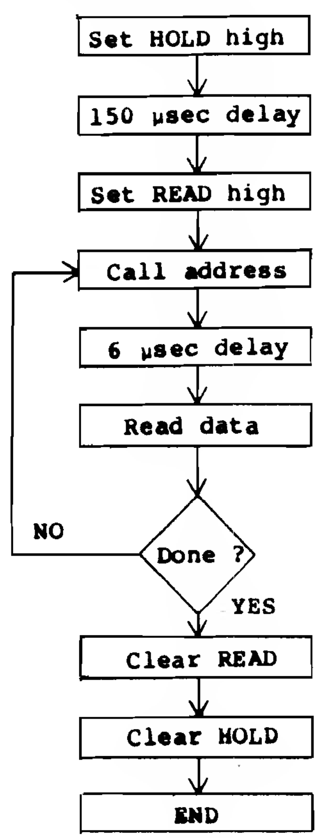


Figure 2: Read Operation

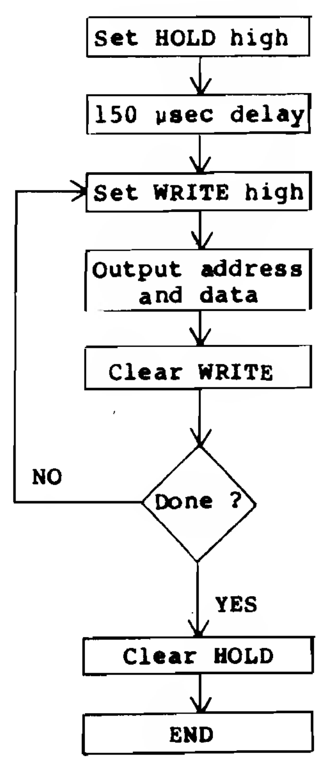


Figure 3: Write Operation

functions except the counting circuits, allowing very low power consumption while still keeping time by the battery backup.

Hold

A high on this line keeps the seconds counter from being incremented by the 1Hz clock output. After the initial set up time (150 microseconds), all counters will be in a static state, allowing error-free read and write operations as long as the HOLD time is less than one second. Other clock circuits do not have this feature, and operations have to be done twice and compared to assure no error has been made.

Consider the following example with a conventional clock circuit. Suppose you are reading a time of 12 hours and 59 minutes. If the seconds count should be 59, and after the hours (and before the minutes) are read, the seconds counter clears and sends a carry pulse, the time is then 13 hours, 0 minutes and 0 seconds. But the read operation has resulted in 12 hours, 0 minutes and 0 seconds—a full hour off. It is for this reason that with conventional clock circuits two reads have to be made to insure proper information has been received.

Read

This input, when taken to Vcc, signals a read operation.

Write

This input, when taken to Vcc, signals a write or set operation. This method of being able to directly set the time is far easier to use than conventional circuits in which pulses must be directed to either a fast set or slow set input, and the clock must read between pulses until the desired time has been set.

±30 Adjust

Momentarily taking this input to Vcc will reset the seconds count to zero. If the seconds count was 30 or more before this action, a carry is sent to the minutes counter. If less than 30, the minutes count remains unchanged. This means that keeping the time accurate is a very simple matter. If the switch in figure 1 is momentarily pressed at the start of a minute, this will automatically reset the time to the correct value as long as the clock is less than 30 seconds either fast or slow.

Test

This input allows testing of the operation of the clock. Pulses to this input will directly clock the S1, MI10, W, D1, or Y counters, depending on which one is addressed by A0-A3.

Reference Signal Outputs

Outputs are available from D0-D3 when READ, CS, and A0-A3 are at Vcc. These can be used as interrupts to the microprocessor. Table 2 presents the conditions for these signals.

Operation

Figures 2 and 3 present the flow diagrams for read and write operations. Although self-explanatory, there are several aspects of the operations which should be emphasized, especially for applications other than the specified one given in this article. First, the HOLD control must always be given at least 150 microseconds set up time, and ***must*** be used for WRITE operations. Next, since the read access time

***Listing 1:* Machine Language Routines**

```
                   Machine language routines for MSM5832
                   Clock/Calendar circuit
                   Randy Sebra July, 1980

                 ACCESS EQ   $8B86  Un-write protect system RAM
                    ORB EQ   $A800  Output register B
                   IORA EQ   $A801  Input/output register A
                   DDRB EQ   $A802  Data direction register B
                   DDRA EQ   $A803  Data direction register A
                    PCR EQ   $A80C  Peripheral Control Register

                        ORG  $0FA7  Start of routines

                   Routine to configure Port B and
                   set HOLD high

0FA7-   20 86 8B SETUP  JSR  ACCESS  Remove write protect
0FAA-   A9 3F           LDA  #3F     Set up PA0-PA5 as outputs
0FAC-   8D 03 A8        STA  DDRA    for address and control
0FAF-   A9 0C           LDA  #0C     Set CA2 low for high
0FB1-   8D 0C A8        STA  PCR     input to HOLD
0FB4-   A0 25           LDY  #25     Delay 150 microsec for
0FB6-   88       DELAY  DEY          HOLD time set up
0FB7-   D0 FD           BNE  DELAY
0FB9-   60              RTS          Return

                   Read routine

0FBA-   20 A7 0F READ   JSR  SETUP   Set up HOLD
0FBD-   A9 00           LDA  #00     Configure PB0-PB3 as data
0FBF-   8D 02 A8        STA  DDRB    inputs for read
0FC2-   A2 0C           LDX  #0C     Initial address
0FC4-   8A       RDLOOP TXA          Transfer to accumulator and
0FC5-   09 10           ORA  #10     combine with READ high
0FC7-   8D 01 A8        STA  IORA    Issue READ
0FCA-   EA              NOP          Small delay
0FCB-   EA              NOP          for read access time
0FCC-   EA              NOP
0FCD-   AD 00 A8        LDA  ORB     Read data
0FD0-   29 0F           AND  #0F     Mask off high 4 bits
0FD2-   95 E9           STA  E9,X    Store to Page Zero
0FD4-   CA              DEX          Decrement address
0FD5-   10 ED           BPL  RDLOOP  Loop until through
0FD7-   A9 0E           LDA  #0E     Then set HOLD low
0FD9-   8D 0C A8        STA  PCR     by CA2 high
0FDC-   60              RTS          Return

                   Write routine

0FDD-   20 A7 0F WRITE  JSR  SETUP   Set up HOLD
0FE0-   A9 0F           LDA  #0F     Configure PB0-PB3 as
0FE2-   8D 02 A8        STA  DDRB    data outputs for write
0FE5-   A2 0C           LDX  #0C     Set initial address
0FE7-   B5 E9    WRLOOP LDA  E9,X    and fetch data
0FE9-   8D 00 A8        STA  ORB     Write data
0FEC-   8A              TXA          Combine address with
0FED-   09 20           ORA  #20     WRITE high and
0FEF-   8D 01 A8        STA  IORA    issue write
0FF2-   29 0F           AND  #0F     Toggle WRITE control
0FF4-   8D 01 A8        STA  IORA
0FF7-   CA              DEX          Decrement address
0FF8-   10 ED           BPL  WRLOOP  Loop until through
0FFA-   A9 0E           LDA  #0E     Set CA2 high for
0FFC-   8D 0C A8        STA  PCR     low on HOLD
0FFF-   60              RTS          Return
```

of the chip may be as long a 6 microseconds, a delay must be built in before reading data. Additionally, notice that although the READ control may be held high for as many read operations as desired, the WRITE control must be pulsed between each write operation.

Interfacing

There are many ways which the MSM5832 can be interfaced with a 6502 or other microprocessor. The only requirement is eleven I/O lines, with four being bi-directional. For myself, the most convenient method was through the use of the #2 6522 VIA on my SYM-1. Figure 1 shows this configuration. If you do not have a 6522 available on your system, it is a relatively simple matter to add one. See "An Additional I/O Interface for the PET", by Kevin Erler, MICRO, December 1979 (19:40). This is also applicable for Apple.

Transistor *Q1* in figure 1 at the HOLD pin is used to invert the CA2 input to HOLD. The reason for this is as follows. On power up and reset, all registers in the 6522 are cleared. This causes all I/O lines to be configured as inputs with a high voltage on the pins, and the HOLD would be held high. When using a battery back up, this would cause the clock to stop until the HOLD is pulled low, since the hold time would always be longer than one second. With the HOLD being controlled separately by the CA2 output and inverted, this will always keep HOLD low unless intentionally taken high by software.

For battery back up, the chip select is connected to +5V from the 6502 bus, which disables all inputs and outputs when the system is off and the clock is on back up. The batteries used here are dry cells and the setup is a rather simple battery back up. A more elaborate setup could be used with NICADs and with the +5V trickle charging the batteries when the system is up. This could give many years of continuous operation before having to replace batteries. The battery life in both cases, of course, is a function of how frequently (or infrequently) the main system is used.

Listing 2: **Basic Routine and Sample Run**

```
 100 REM
 110 REM      MSM5832 CLOCK/CALENDAR
 120 REM      SET/READ PROGRAM
 130 REM      RANDY SEBRA  JULY,1980
 140 REM
 150 DEF FNS(X)=INT(X/10)
 160 DEF FNT(Y)=Y-FNS(Y)*10
 170 INPUT"SET(S) OR READ(R) ? ";I$
 180 IF I$<>"S" THEN 430
 190 REM
 200 REM      GET INPUT AND STORE INTO LOCATIONS $E9-$F5
 210 REM
 220 INPUT"MONTH,DAY,YEAR(2 DIGITS) ? ";M2,D,Y
 230 POKE 245,FNS(Y)
 240 POKE 244,FNT(Y)
 250 POKE 243,FNS(M2)
 260 POKE 242,FNT(M2)
 270 POKE 241,FNS(D)
 280 POKE 240,FNT(D)
 290 INPUT"DAY OF THE WEEK(1-7) ? ";W
 300 POKE 239,W-1
 310 INPUT"HOURS,MINUTES(24 HOUR TIME) ? ";H,M
 320 POKE 238,FNS(H)+8
 330 POKE 237,FNT(H)
 340 POKE 236,FNS(M)
 350 POKE 235,FNT(M)
 360 REM
 370 REM      CALL MACHINE LANGUAGE WRITE ROUTINE
 380 REM
 390 S=USR(&"0FDD",0)
 400 REM
 410 REM      CALL MACHINE LANGUAGE READ ROUTINE
 420 REM
 430 R=USR(&"0FBA",0)
 440 DIM D$(6),M$(11)
 450 DEF FNR(I)=PEEK(I)*10+PEEK(I-1)
 460 DATA SUNDAY,MONDAY,TUESDAY,WEDNESDAY,THURSDAY,FRIDAY,SATURDAY
 470 DATA JANUARY,FEBRUARY,MARCH,APRIL,MAY,JUNE,JULY,AUGUST,SEPTEMBER
 480 DATA OCTOBER,NOVEMBER,DECEMBER
 490 FOR I=0 TO 6
 500 READ D$(I)
 510 NEXT I
 520 FOR I=0 TO 11
 530 READ M$(I)
 540 NEXT I
 550 REM
 560 REM      CONVERT PAGE ZERO DATA INTO APPROPRIATE UNITS
 570 REM
 580 Y=FNR(245)+1900
 590 M2=FNR(243)
 600 D=FNR(241)
 610 W=PEEK(239)
 620 REM
 630 REM      CONVERT HOURS, MINUTES, AND SECONDS INTO A
 640 REM      STRING FOR "CLEANER" OUTPUT"
 650 REM
 660 T$=""
 670 FOR I=0 TO 4
 680 T$=T$+RIGHT$(STR$(PEEK(237-I)),1)
 690 NEXT I
 700 T$=RIGHT$(STR$(PEEK(238)-8),1)+T$
 710 T$=LEFT$(T$,2)+":"+MID$(T$,3,2)+":"+RIGHT$(T$,2)
 720 PRINT"TODAY IS ";D$(W);" ";M$(M2-1);D;Y;T$
 730 END
OK
RUN
SET(S) OR READ(R) ? S
MONTH,DAY,YEAR(2 DIGITS) ? 7,18,80
DAY OF THE WEEK(1-7) ? 6
HOURS,MINUTES(24 HOUR TIME) ? 13,8
TODAY IS FRIDAY JULY 18  1980 13:08:00

OK
RUN
SET(S) OR READ(R) ? R
TODAY IS FRIDAY JULY 18  1980 13:09:10
```

Machine Language Routines

Listing 1 presents machine language routines to set the clock/calendar and read the time. As shown by figure 1, PA0-PA3 go to address lines A0-A3, PA4-PA5 go to the READ and WRITE, PB0-PB3 go to data lines D0-D3, and CA2 is inverted and goes to the HOLD input.

Data to be written to the clock, and the data received from a read are stored in Page Zero locations $E9-$F5. These are "safe" Page Zero locations which are not used either by BASIC nor by the SYM monitor. For computers other than the SYM, other locations may have to be used, but virtually all 6502 computers will have Page Zero locations available.

The routines themselves are general routines which may be used for any 6502 computer since they do not use any monitor routines, except the routine necessary to remove the write-protect from system RAM. Of course, the locations for the 6522 will probably be different. The machine language is located so as to occupy the highest part of memory in a 4K system. They can be easily relocated, with the only changes required being the JSR's at locations $0FBA and $0FDD in listing 1.

Applications

The obvious use for the clock/calendar interface is setting the time and getting the time output upon request. Using the machine language routines in listing 1 in conjunction with a BASIC driver is perhaps the most convenient method of accomplishing this. Listing 2 is an example of one such BASIC program, along with two typical resultant runs. The program, in order to set the clock/calendar, merely requests the necessary data and stores it into the proper Page Zero locations and then calls the machine language routine to do the actual setting. To insure that the input data was correct, a read is done after the setting as a check. For the read operation, the program calls the machine language to do the actual reading, and then merely arranges the data obtained from $E9-$F5 to be output in a convenient manner.

The memory size of 4006 is for a 4K system. The dummy tape save, SAVE D, is needed to overcome a bug in SYM BASIC. The program is loaded in as file "C". The machine language routines were saved as file $4D, so that they can be loaded by the LOAD M command.

***Listing 3:* Interrupt Set-up Routines**

```
                     Machine language routines for
                     Setting up interrupts from the MSM5832
                     Randy Sebra July, 1980

                   ACCESS EQ   $8B86  Un-write protect system RAM
                      ORA EQ   $A801  Input/output register A
                     DDRB EQ   $A802  Data direction register B
                     DDRA EQ   $A803  Data direction register A
                      IFR EQ   $A80D  Interrupt flag register
                      IER EQ   $A80E  Interrupt enable register

                          ORG  $0F8C  Start of routine

0F8C-   20 86 8B SETUP    JSR  ACCESS  Remove write protect RAM
0F8F-   A9 00             LDA  #00     Set up B0-B3 (data
0F91-   8D 02 A8          STA  DDRB    lines) as inputs
0F94-   A9 1F             LDA  #1F     Set up A0-A4 (address
0F96-   8D 03 A8          STA  DDRA    lines and READ) as
0F99-   8D 01 A8          STA  ORA     outputs and set all high
0F9C-   A9 08             LDA  #08     Set up IFR for interrupt
0F9E-   8D 0D A8          STA  IFR     from CB2
0FA1-   A9 88             LDA  #88     Enable interrupt for
0FA3-   8D 0E A8          STA  IER     CB2
0FA6-   60                RTS          Return
```

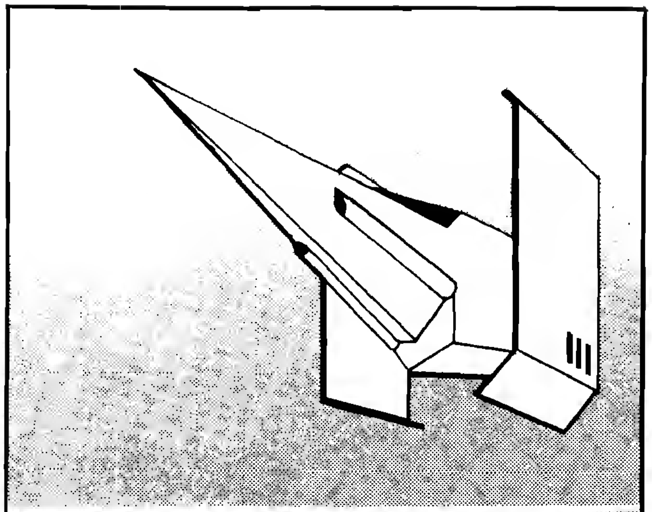
As mentioned previously, the clock can generate interrupts to the microprocessor. Since we are using a 6522 VIA in the interface, either of the two on-board timers can be used to generate precise interrupts of up to 0.65 seconds apart. With the MSM5832, we then can generate interrupts at one second, one minute, or one hour apart (see figure 1 and table 2).

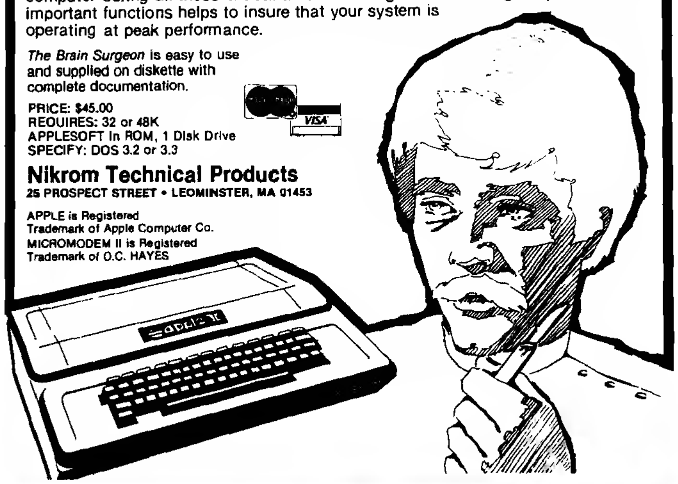
Listing 3 presents a machine language routine to use any one of these three interrupts. Although the D1, D2 or D3 outputs could be tied directly to the $\overline{IRQ}$ line of the 6502 system, in this example one of the outputs goes to the CB2 input of the 6522. The routine has been set up so that the interrupt occurs on the negative-going edge of the 122.1 microsecond pulse. If setting on the positive edge is desired, merely write a $20 in the PCR register of the 6522. In whatever interrupt routine used with this setup, the IFR register bit can be cleared, either by directly writing into the IFR, or by reading or writing to Port B. This may be accomplished by reading the time with the routines in listing 1. Note, however, that reading the time reconfigures Port A, and this must be reset to the configuration in listing 3.

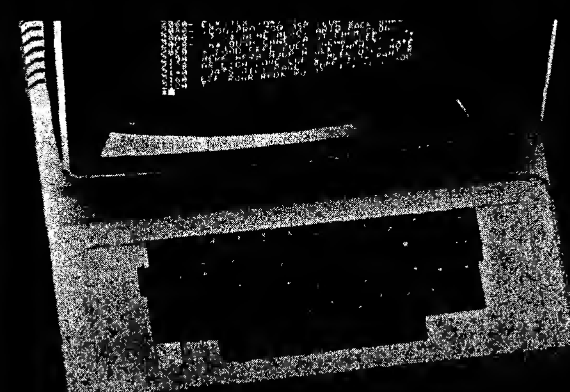
An obvious use of this type of operation would be to use an interrupt of one second when using a video display or terminal with an addressable cursor, to continually write the time in the upper right hand corner of the screen. A much more effective use would be in a polled environment where it would be desirable to get data from input ports or status of peripherals every second, minute, or hour.

Special note: The MSM5832 is a fairly new chip, first introduced in the first quarter of 1980. For this reason, it is not commonly available, except in quantity from the manufacturer. If there is sufficient reader interest in using the circuit described in this article, I can supply the chip and the 32.768 KHz crystal for $17 plus postage. Delivery may take 4 to 6 weeks.

MICRO







This February issue of the Ohio Scientific Small Systems Journal consists entirely of an introduction to the world of Artificial Intelligence (AI). We hope that you enjoy it.

The Use of Microcomputers in Artificial Intelligence Research

INTRODUCTION

In the first issue of *AI Magazine* (see reference 1), a publication of the newly-founded American Association for Artificial Intelligence (AAAI), Artificial Intelligence is defined as "that part of Computer Science concerned with the symbol-manipulation processes that produce an act or decision that is goal-oriented, arrived at by an understandable chain of symbolic analysis and reasoning steps, in which knowledge of the world informs and guides the reasoning."

A simpler definition is offered by Phillip C. Jackson in his book *Introduction to Artificial Intelligence* (see reference 2): "Artificial Intelligence is the ability of machines to do things that people would say require intelligence." This definition does not specify how to tell that a machine has intelligence on a human level, but that problem was solved nicely by A.M. Turing in 1947 (see reference 3). In Turing's classic test, a human interrogator is allowed to question two sources, one human and one machine, on a particular topic of intellectual endeavor. The responses of the two sources are presented through a common, neutral medium such as a teletype, to mask their origin. If the interrogator is unable to determine which source is responding (with accuracy significantly greater than 50%) and this result continues to hold as the experiment is repeated with various human sources and human interrogators, then the machine has exhibited artificial intelligence.

Now, there is widespread and justifiable doubt that a machine will ever exhibit human-like intelligence in a general sense. But this is not required by the preceding definitions. In Turing's test the interrogation takes place in a particular, restricted area of human endeavor. If a machine passes Turing's test, then it has simulated human intelligence, but only in this restricted area. In some areas there has been a good deal of success. But in other areas, natural language, for example, machine simulation of even a child's ability seems to be extremely difficult. Therefore, rather than debate the existence of artificial intelligence in general, it is more fruitful to concentrate on particular aspects of human intellectual ability.

When an artificial intelligence research project seeks to program a machine to imitate expert human behavior in a specific intellectual area, it is called knowledge engineering. The specific area is called the task domain of the project. In the next section, we will survey some of the most active task domains. The discussion there is intended to be illustrative, not comprehensive. We will concentrate below (see *Natural Language Processing*) on the very complex task domain of natural language. The final section contains a description of recent progress made at Ohio Scientific on the development of language understanding programs for Ohio Scientific microcomputers.

EXAMPLES OF AI TASK DOMAINS

Writing a program that plays "perfect" Tic-Tac-Toe is a relatively simple exercise. The goal states or winning positions in this game are relatively few and the number of ways a player can reach one of the winning positions is also quite limited. The program can be constructed so that the computer makes each of its moves based on a complete analysis of all possible moves.

Unfortunately, most task domains are far more complicated than Tic-Tac-Toe. Games such as chess simply have too many states to make feasible a brute force approach—one that requires sequential consideration of all possibilities. Other activities of the human mind, such as theorem-proving, visual perception, and natural language processing, involve cognitive processes that are only partially understood. Here, AI research goes hand in hand with work in cognitive psychology. As we learn more about how humans resolve ambiguities, fill in missing details, and make judgments and decisions on intuitive levels, we will be better able to create computer simulations of these mental processes. This will surely be enormously difficult, but it would be a mistake to be overly pessimistic about the limitation of artificial intelligence. If the human mind possesses awesome powers that we are just beginning to understand, then it probably has the power some day to train a computer to do things that are presently unimaginable. Indeed, in many task domains, AI researchers have made progress that has greatly exceeded what was thought possible only a generation or so ago. This progress can be attributed partly to advances in hardware and partly to advances in the modelling of cognitive processes.

A good example is the task domain of chess. Just twenty years ago, the fact that computers could not be programmed to play any better than a beginner's level was used by many as "proof" of the limitations of artificial intelligence. The number of different chess games that could be played, approximately 10^{120} (see reference 4) was thought to be an impossibly large number. Today, of course, there are many outstanding chess-playing programs available (a very good one, SARGON II, is available for Ohio Scientific computers). There are several competitions for chess-playing programs and the winners usually play at close to championship levels. Sophisticated search techniques, that reduce the number of moves the computer must consider and techniques that allow the computer to "learn" from previous mistakes, are generally features of these programs.

Another active task domain is problem solving and decision making. One of the earliest programs was the classic General Problem Solver (GPS) of Newell, et al. (see reference 5). This program, with

later refinements, was able to solve a wide variety of problems that had a well-defined goal or solution state and list of actions that could possibly modify present states. This restriction on the kind of problems that GPS would accept is an example of the modelling assumptions that must be made on how a human thought process takes place before it can be computer-simulated. The program SAINT is an example of a symbolic manipulation problem solver. More specifically, it solves integration problems at about the level of a first-year calculus student.

Pattern perception is the task domain in which we seek to make order out of apparent disorder. The kinds of patterns we hope to discern might be based on sounds, symbols, or even forms of reasoning, but artificial intelligence has been especially concerned with visual patterns (see reference 6). One of the most important applications is the development of machines with limited "reading" ability (e.g., the ability to read zip codes for the postal service). Another area of current activity is the development of special-purpose remote visual sensors ("seeing-eye robots") for certain medical procedures and industrial operations. In April of 1981, the first international conference on Robot Vision and Sensory Controls (ROVISEC) will be held.

The final task domain we will discuss is that of natural language. There are really two aspects to consider. The first, spoken language, involves a decoding, or pattern-matching, of sound to language symbols (voice input) or the generation of sound from language symbols (voice output). Much progress has been made on voice output. Ohio Scientific currently offers a complete package, that includes the Votrax voice synthesizer, that is useful for experiment or development in this area. Because of the range of human voices and language dialects, the problem of voice input is much more complicated. One way of simplifying this problem is to restrict greatly the context of the voice input message. For example, systems are currently being developed that will "understand" a user's spoken request for airline flight information or reservations.

The second aspect of natural language is written language. Here we assume we have the actual specific language representation of a message and we seek somehow to understand it and properly respond. In the next section, we will focus on the difficulties faced by artificial intelligence researchers who try to develop computer simulations of this process.

NATURAL LANGUAGE PROCESSING

Complexity of the General Problem

There are two extreme opinions on the solution of the natural language problem. The first is that it can't be done, usually based on the premise that human intelligence is so complex that we'll never fully understand it. The second attitude is that it can be done, and the only problem is to string together enough hardware and software.

Most probably, neither of these extreme views is correct. The more likely and a somewhat more reasonable view is that certain aspects of natural language processing can be accomplished by a computer. Note that the word "processing" has been used rather than "understanding" or "comprehension." This more neutral term avoids ascribing consciousness to machines and comparing it with human consciousness.

In the past ten years much progress has been made in the field of artificial intelligence. Several systems have been demonstrated (see Green's and Lindsay's papers, reference 7; also, reference 8) which *will* handle natural language input. Many theoretical systems have been suggested to process the large corpus that is natural language. While the theoretical systems exist, only small portions of them have been implemented to any degree. Winograd's theoretical system for understanding natural language (see reference 9) is relatively complete but it has been demonstrated only for a small block manipulation environment. There are a number of similar examples from major research centers operated by Stanford University, Carnegie-Mellon University, Massachusetts Institute of Technology, Bolt, Baranek, and Newman, Inc., and International Business Machines Corp. Why is it that the accomplishments of those interested in natural language processing have not matched their ambitions? It is certainly not due to lack of resources or talent. Rather it can be attributed to the enormous complexity of the task. There is a finite set of words, but the set of sentences that can be generated is theoretically if not practically infinite. Even with severe restrictions on vocabulary size and grammatical form, the problems of natural language processing are still substantial.

Rather than attempt to solve the whole problem, which is the usual approach, the approach used at Ohio Scientific has been to construct a system which would handle a small domain of natural language usage. This approach allows us to deal with a smaller vocabulary and to analyze a smaller number of syntactic forms. While semantic analysis remains a complex problem, response generation, the ultimate test of success, can be evaluated in this limited context.

The following sections will discuss the construction of characteristics of the dictionary or vocabulary, the syntactic or parsing analysis, the semantic analysis, and finally the response generation.

Construction of a Limited Understanding System

The Dictionary: The task of the dictionary is to provide a vocabulary which will form the basis for a series of functions that the machine will perform. The bulk of a dictionary should consist mainly of nouns and verbs, the content words. The selection of adjectives and adverbs depends upon the particular domain that is to be considered.

While the vocabulary may be of limited size, it is functionally much larger, because many words have multiple usages. The most difficult category to select is that of verbs. Frequency of usage is the easiest criterion to employ. Since many verb

forms are irregular, care must be taken to include these as separate entries. All the common modal or linking verbs can be included, even in a vocabulary of limited size. Pronouns, adverbs, prepositions, and conjunctions are relatively few in number. Thus, all the common ones can be included.

The dictionary developed for the Ohio Scientific programs satisfies all of the above conditions. It also has the following features:

Grammatical Usage and Function: Each word entered has a part of speech tag and a function tag attached. The first tag indicates the part of speech. The second tag indicates its usage. For example, the noun JOHN would be classified as Animate, Person, Proper Name. While the word CUSTOMER would be classified as Animate, Person, Generic.

Verb classifications include information as to whether the verb is modal (could, would) or linking. Regular and irregular forms are distinguished as are verbs of action, cognition, identity and part-whole relations. A tense indicator is included for irregular forms. The part of speech and function information will be used in the parsing section of the program.

Multiple Usage: One characteristic of English is that one word may serve as several different parts of speech. For instance, the word AVERAGE can be a noun, a verb, or an adjective. Compute the AVERAGE, AVERAGE these numbers, and AVERAGE rainfall show these multiple usages.

Words that have multiple usages are given additional tags. The word AVERAGE has three different tags. Where possible, these tags are in terms of frequency of usage.

Root Word Approach: For the most part, only root words appear in the dictionary. Plural forms, possessive forms, tense forms are determined by information derived from a "snipping" routine. This approach accomplishes two things. First, the vocabulary can be kept relatively small. Second, the information from the ending can be used to determine the part of speech and function that is to be applied to the root word.

Irregular verb forms and certain contractions require a separate dictionary entry. Separate entries are also required for irregular forms of adjectives.

It is not possible to develop a list of functions for all the words that lead to an exhaustive set of categories. A less than exhaustive category list results in certain anomalies. If the words WEAK and GREEN are treated as adjectives indicating quality, then there may be no provision for determining the illegal form GREENLY.

As the vocabulary is currently constituted, it is relatively easy to add new entries. If new entries involve no modification of the category identifiers, they can be inserted into the dictionary with little difficulty. Words currently in the dictionary can be deleted or their tags can be altered to fit current usage.

Syntactic Analysis (Sentence Parsing): The goal for a sentence parsing program is to decompose the string of words or phrases and to determine the relationships between these various parts. The most popular approach to this task in the psycholinguistic literature has been the transformational grammar approach of Chomsky (see reference 2). This method involves taking all forms of a sentence and applying transformational rules to yield the basic or kernel sentence. This approach, while simple to describe, is difficult to do. Also, several authors (see references 8 and 10) have suggested that the transformational approach is more suited to sentence production than recognition. There are also several other approaches to syntactic analysis.

Word Serial Position: The ordering rules in English are not as regular as in many languages. For most simple dialogue, word serial position can give much information about the relationships between words. The Simple Active Declarative (SAD) form usually consists of Noun-Verb-Noun. Word serial position, along with the part of speech and function tag, can be used to assign sentences to various general types. For example, a verb in position one usually signals an imperative mode. A WH pronoun (e.g., Who, What, Where, Why) in position one usually signals a question.

Word serial position and the grammatical tags may be used to develop a network to determine permissible and non-permissible sentences. This could also be used to assign the program to the appropriate response mode.

Phrase Boundary Analysis: This approach to parsing attempts to divide the sequence of words into its constituent structure, primarily noun and verb phrases. The constitutent structure is then represented by a tree diagram or a series of brackets showing the relationship between the constituent parts. The classical example is the sentence "They are eating apples" which has two distinct meanings. Constituent analysis allows one to determine how structure points to one or the other.

(They^{NP}) (are eating^{VP}) (apples^{NP}.)

(They^{NP}) (are^{VP}) (eating apples^{NP}.)

The advantage of the phrase approach is that it deals with larger units than words and can eliminate a number of parsing steps that add little information. The disadvantage of this approach is that many sentences cannot be disambiguated using this approach. For example, the sentence "Flying planes can be dangerous" does not parse out into the two possible meanings.

Examples like the above are used to point out the weakness of a phrase boundary approach but the response is that this sentence would be clearly perceived if the context were known.

Clark and Clark (see reference 9) provide some strategies for determining the phrase structure of a sentence. These rules are relatively clear cut and can be adopted into a parsing program. As with the Word Serial Order, there are relatively stable patterns that occur in the constituent structure, and these can yield information about sentence type.

Sentence Frame: A good deal of information about sentence structure can be gathered from the endings that are attached to words. A sentence frame approach makes use of this information and the position of the function words.

The _______s have ______en the _______s

The foxes have eaten the chickens

This frame can be used to determine that the second word is a noun and the fourth word a past participle and the last word a noun used in the objective case.

Another approach that would utilize the sentence frame would be to limit the types of frames that would be accepted. This might be a good approach for a small demo project, but it obviously limits the generality of the program.

osi cont again
1st run

Augmented Transition Networks: One approach developed in the early 70's sought to reduce the complexity by using a system called Augmented Transition Networks (ATN). The basic task of the ATN is to reduce the sentence to a set of relationships indicating Action, Actor, and Object. Augmented Transition Networks can handle conditional statements, relative clauses, and passive transformation.

The analysis of a sentence using ATN's proceeds in a left to right fashion. The contents of each register can change as the analysis proceeds. Thus, each register has a state description, but this description can change the state of another register. For the example of "The man was bitten by the dog," the initial state of man as actor is changed to object as a result of the detection of "by the dog." The ATN approach has sufficient strength to handle alternative sentence structures but, at the same time, is simple enough to be processed quickly.

One of the most complete models that has used the ATN approach is one developed by Winograd (see reference 8). The demonstration part of the system consists of an artificial robot whose response to commands is shown on a video display. The robot manipulates a number of objects with different shapes and can report upon their status. The program is written in LISP and runs on a PDP-10 and requires 80K of core. The system consists of a dictionary, grammar, semantic analyzer and several specialized subprograms such as Planner and Programmer. The system accepts typed input and carries out commands and answers questions in 5-20 seconds.

Semantic Analysis: The previous section on syntactic analysis has focused on the decomposition or parsing of a sentence. The next step is to use this information to determine the semantic relations or the "meaning" of the sentence.

Implicit in any approach to semantic analysis is the concept of a knowledge base. The form and use of the knowledge base are particularly sticky problems. Let us use a particular example to introduce this topic. Suppose we input the question, "Is John taller than Mary?" This sentence could be paraphrased as "Male person greater (height) than female person?" Even a more general form might be "Object A possesses more (quantity) Object B." If there is specific information about John and Mary in our knowledge base, the question can be easily answered. If our knowledge base contains only a fact that shows males are usually taller than females, an inference can be drawn in response to the question. If Object A and Object B have some attributes in common, a comparison can be made and a response derived. The crucial question then becomes how to structure the knowledge base. This is a very complicated problem that is intertwined with the problem of response generation.

Response Generation: ELIZA (see reference 2), a program that simulated a psychotherapist, solved the response generation problem quite easily by simply responding "that's interesting; tell me more", "I understand" or something similar whenever the input sentence contained none of a limited number of key words. But any practical language processing system (including human beings) cannot respond meaningfully to every input sentence. Basically, response generation involves a mapping of the meaning of the input sentence to some capability of the machine. Thus, a declarative statement such as "I ran three miles today" might be "understood" quite thoroughly by a language processor but no meaningful response can be generated unless, for example, the program also has the capability of storing the input sentence's content in, say, the exercise log of the user's personal health data base.

Next month, we will expand upon the response generation problem in a description of experimental language processing work undertaken at Ohio Scientific.

References

1. *AI Magazine,* American Association for Artificial Intelligence, 5147 Angeles Crest Highway, LaCanada, California 91011.
2. *Introduction to Artificial Intelligence,* by Phillip C. Jackson, Mason/Charter Publishers, 1974.
3. "Intelligent Machinery," by A.M. Turing (1947); reprinted in *Machine Intelligence 5,* edited by B. Meltzer and D. Michie, 1970.
4. "Programming a digital computer for playing chess," by C.E. Shannon, *Philosophy Magazine,* 41 (1950), pp. 356-375.
5. "GPS, a program that simulates human thought," in *Computers and Thought,* edited by Feigenbaum and Feldman, McGraw-Hill, 1963.
6. *Pattern Classification and Scene Analysis,* by R.O. Duda and P.E. Hart, Wiley, 1973.
7. *Computers and Thought,* edited by Feigenbaum and Feldman, McGraw-Hill, 1963.
8. *Understanding Natural Language,* by Terry Winograd, Academic Press, 1972.
9. *Psychology and Language: An Introduction To Psycholinguistics* by Clark and Clark, Harcourt, Brace and Jovanovich, 1977.
10. "Transition Network Grammars for Natural Language Analysis," *Communications of the ACM,* 1970, 13, pp. 591-606.

PET and APPLE II Users

PASCAL

ABACUS Software makes available its version of TINY PASCAL for the users of two of the most popular personal computers.

TINY PASCAL is a subset of the standard PASCAL as defined by Jensen and Wirth. It includes the structured programming features: IF-THEN-ELSE, REPEAT-UNTIL, FOR TO/DOWN TO-DO, WHILE-DO, CASE-OF-ELSE, FUNC and PROC. Now you can learn the language that is slated to become the successor to BASIC.

TINY PASCAL is a complete package that allows you to create, compile and execute progams written in the PASCAL language. You can save source and object code on diskette or cassette (PET version only). Comprehensive user's manual included. The manual can be examined for $10 (refundable with software order).

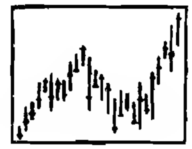
REQUIREMENTS

PET 16K/32K New ROMS cassette	$40
PET 16K/32K New ROMS diskette	$35
Apple II 32K Applesoft ROM w/DOS	$35
Apple II 48K Applesoft RAM w/DOS	$35
TINY PASCAL User's Manual	$10
6502 Interpreter Listing	$20

FREE postage in U.S. and CANADA
All orders prepaid or COD

ABACUS SOFTWARE
P. O. Box 7211
Grand Rapids, Michigan 49510

MARKET CHARTER(tm)



STOCK CHARTING ON YOUR APPLE II*

Market Charter. $129.95

- High-Low-Close Bar Charts
- Simple, Exponential, Weighted Averages
- Trendlines, Resistance Lines, etc.
- Volume Charts with average volume
- Hard Copy of the Charts and Data
- Comparison Charts
- Weekly & Daily Stock Histories available
- User Oriented
- User can create and update the Data base
- Many Satisfied Users

DowLog-MC. $69.95

- Automatically update data
- Quick, easy & inexpensive

RTR SOFTWARE, INC.
Dept. M-1
1147 BALTIMORE DR.
EL PASO, TEXAS 79902
(915) 544-4397

*TRADEMARK OF APPLE COMPUTER, INC.

Having Trouble Finding KIM & AIM Software?

LOOK NO MORE!

MICRO sells software packages that may be just what you have been searching for!

All products come complete with full user/operator instructions, a cassette tape and, with the exception of MICROADE, a complete set of source listings.

KIM	CASSETTES/ MANUALS
KIM Microchess	$15.00
HELP Mailing List	15.00
HELP Info Retrieval	15.00
MICROADE	25.00
AIM	
Microchess	15.00

To place your order, write or call:

MICRO
P.O. Box 6502
Chelmsford, MA 01824
(617) 256-5515

Dealer Inquiries Invited

THE MAILING LABEL AND FILING SYSTEM

From Avant-Garde Creations **only $24.95 ppd.**

This unique system will handle both your filing needs and your mailing label needs.

It's uniqueness starts with user-determined variables (up to 10 options) and continues with a special COUNT/SORT routine that allows the user to sort up to 9 VALUES for each of any 9 (out of 18) variables. It will print mailing labels, do a regular print-out or just display the criteria-meeting records while it counts them. It will also range-sort for 3 particular variables.

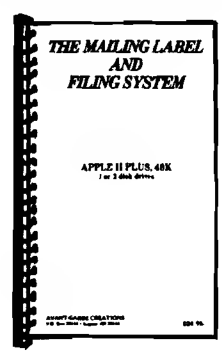
It makes an alphabetized directory of names and record numbers. You can find records by name or by numbers in seconds. If you don't know the exact spelling there's a quick-find option for directory-reading.

You can customize your labels and print up to 6 lines of your variables on them.

It includes special quick-copy and backup programs.

An easy to use system, brimming with options and dynamics, which ends the need for separate filing and mailing label programs.

Includes 40-page program manual and disk. APPLE II PLUS, 48K, one or two disk drives.



$24.95 ppd.
We accept VISA/Mastercharge

Write for our detailed brochure and more information:

Avant-Garde Creations
P.O. Box 30161
Eugene, OR 97403
(503) 345-3403
noon until 6:00 pm
Dept. m

MICRO Software Catalog: XXIX

Name: **What's That Song?**
System: Apple II or Apple II Plus
Memory: 16K
Language: Machine
Hardware: No special hardware required.
Description: The Apple II begins playing a song from its repertoire of well-known melodies using its built-in speaker. The melodies included cover a wide range of musical selections. The computer asks the first player who recognizes the song to type in the title. The program has a unique comparison algorithm that eliminates the necessity of typing the song title exactly; but it will not accept just anything! It also keeps score for one to three players. The Apple knows 151 songs; how many do you know?
Price: $24.95 includes cassette, operating instructions.
Author: **Daniel J. Hughes**
Available: **JDeL LectricWare, Inc.**
P.O. Box 9140
St. Louis, MO 63117

Name: **Applesoft Renumber/ Merge in ROM**
System: BASIC Apple II or Apple II Plus
Memory: 48K
Language: Assembly (ROM Chip)
Hardware: Mountain Hardware's ROMPLUS Board
Description: Plug this ROM into your ROMPLUS board and this utility will be a keyboard command away from your immediate use. This invaluable program, made famous by Apple Computer, needs no explanation for the serious programmer. It will renumber all or part of an existing program with any size increment. It will move portions of a program within itself. It will also hide an existing program to allow another program to be loaded and reviewed or it will merge the two into one. When activated, it will not disturb any part of a program which may already be in memory.
Copies Many
Price: $49.95 includes user documentation and ROM chip
Author: **Frank D. Chipchase**
Available: **Soft CTRL Systems**
P.O. Box 599
West Milford, NJ 07480

Name: **Addres**
System: OSI C8P DF/C3 Series
Memory: 32K
Language: BASIC OS65U
Hardware: 8" floppy disk
Description: Beyond the valley of the OS-DMS address program—extremely quick access—prints labels, finds names by first or last names—DMS compatible file. Also other goodies. Write for details.
Copies Available now
Price: $10 — your diskette; $20 — my diskette
Author: **Blake Etem**
Available: **Blake Etem**
Box 3, Det 4, 40th TACG
APO New York, NY 09161

Name: **6502 Cross Assembler in 8080 Language**
System: SOL or CUTER 8080 with Processor Tech Software #1
Memory: 8K
Language: 8080 Assembler
Description: Permits development of programs for KIM or other 6502 systems too small to support an assembler. Uses editor and many subroutines in the Processor Technology Software #1 self-contained system. Object code may be downloaded to KIM in paper tape or cassette format.
Copies New
Price: Write for Details
Author: **Albert S. Woodhull**
Available: **A.S. Woodhull**
RFD2 33 Enfield Rd.
Amherst, MA 01002

Name: **Board Games 1**
System: OSI C1,C2,C4,C8 BASIC-in-ROM
Memory: 8K
Language: BASIC and Machine
Hardware: None special
Description: Board Games 1 consists of two games: Cubic and Mini-Gomoku. Cubic, written in BASIC, is a game of three-dimensional tic-tac-toe, with graphics. Mini-Gomoku is a gomoku game, written in machine language, also with graphics.
Copies Just Released
Price: $15
Author: **Terry Terrance & Danny Schwartz**
Available: **Orion Software Assoc.**
147 Main Street
Ossining, NY 10562

Name: **Applointments**
System: Apple II or Apple II Plus
Memory: 48K RAM
Language: Applesoft - 3.2.1 DOS
Hardware: Applesoft firmware card, 1 disk drive, optional printer
Description: The Applointments package allows you to create your own appointment system with user definable starting and ending times, appointment separation and entry sizes. Enter, change and erase editing with AM-PM and day to day viewing at a touch of a key. Form messages, activate-deactivate schedule dates. Full search, phone, treatment/meeting code options and many more easy to use features. For the busy professional or businessman. 7500+ appointments per disk.
Copies Just Released
Price: $60 includes manual, tutorial program, disk.
Author: **Guy Lyle**
Available: **Andent Inc.**
1000 North Ave.
Waukegan, IL 60085

Name: **WP6502 V1.2**
System: ALL OSI
Memory: 8K Tape, 20K Disk
Language: Machine
Description: An easy to use word processing system. Editing features include line editing, Global search and replace with echo check, and ability to view text on either the screen or the printer. All commands are a single letter selected from a menu. Embedded commands include changing margins and spacing, both standard and dotted tabs, and Keyboard insertion. Blocks of text, numbered from 01 to 99, can be of unlimited length and may be inserted anywhere in the main text. WP6502 also has a unique AP feature which prevents paragraphs from being broken at the end of a page.
Copies Many
Price: $50 Tape, $100 Disk (5", 8" 65D or U)
Author: **Dwo Quong Fok Lok Sow**
Available: **Dwo Quong Fok Lok Sow**
23 E. 20th St.
New York, NY 10003

Name: **COPY/1**
System: OSI C4P
Memory: 24K
Language: Machine Code
Hardware: One minifloppy drive
Description: An efficient copier that can copy a fully loaded 40 track disk onto a blank disk with only four mountings of each disk, in two minutes from a cold start. Includes track zero, disk initializing, start and end on any tracks, multiple (0-255) copies, track/sector/page count listing following each read or write action. In full color with sound cuing. Needs no printed instructions or typed input.
Copies: Just released
Price: $20 (MD residents add 5% tax) includes diskette and mailing
Author: **Hugh Tornabene**
Available: Box 928
College Park, MD 20740

Name: **Recall**
System: Apple II Plus
Memory: 48K
Language: Applesoft
Hardware: Apple II Plus, Disk Drive, Printer Optional
Description: An Arbitron ratings analysis package for radio stations. Compares up to four radio stations on a single display. Allows printer output and features color bar graphs. The program computes and displays audience turnover, time spent listening, audience re-cycling to day-parts, cume and quarter-hour analysis, and mutual exclusive cumes.
Price: $750, includes manual
Author: **Dr. Roger Skolnik**
Available: **Media Service Concepts**
Box 10682
Chicago, Illinois 60610

Name: **Drill Skill (Math)**
System: Apple II
Memory: 48K (Fits 16K+)
Language: Integer BASIC
Hardware: Disk or Tape
Description: Lo-Res Educational mathematics which randomizes your questions and lets you decide the amount of permissible errors. Program is open-ended so that it can be customized and changed in its operational functions (see REM). When completed it counts correct amount vs. amount of errors.
Copies: On Demand
Price: Disk $10.95, Tape $6.95 plus .50 s/h
Author: **R. Sherman**
Available: **PCS Electronics**
52 Jackson Dr. So.
Poughkeepsie, NY 12603

Name: **Budget Estimator**
System: Apple II or Apple II Plus, printer optional
Memory: 32K
Language: RAM or ROM Applesoft plus machine code
Description: This program allows user to estimate small business operating budgets before they are generated in detail. It is very useful for creating ballpark estimates. The program is based upon entering average pay/employee/period, # of days/-period, non-labor costs # of employees/period, plus entries for extraordinary expenses. Data base can be stored on disk for future recall and comparison with 'actuals'. Supports printer with machine code formatting routines.
Price: $24.95 includes DOS #3.3 diskette, description, plus example.
Author: **Neil A. Robin**
Available: **Tech-Digit Co.**
21 Canter Lane
Sherwood, Oregon 97140

Name: **Coil Design**
System: PET computer
Memory: 6K
Language: BASIC
Hardware: 1.0 or 2.0 ROM PET
Description: Design your own R.F. chokes and tank coils to a specified inductance or inductive reactance on a coil form of your choice. Coil inductance may be from 0.2 to 75 microheneries using wire sizes from #8 to #40 with a special section for tubing. Designs include using ½, 1, or 2 watt resistors as forms, or you may choose a form having dimensions meeting your requirements.
Copies: Just released (SASE for catalog)
Price: $3.95 ppd.
Author: **Harry L. Rosier**
Available: **Kinetic Designs**
401 Monument Rd. #171
Jacksonville, FL 32211

Name: **FORM-DS**
System: Apple II
Memory: 32K, ROM Applesoft
Language: Applesoft
Hardware: Disk
Description: FORM-DS is a system of programs and routines that assist in the entry, editing and display of data. Simplifies the use of sophisticated I/O, in your programs. Describe screen formats by simply typing them on the screen. Automatic range tests of numeric input data. Displays edited numeric values with commas inserted, etc. Dump the text screen contents at any time to a printer. Routines are easily incorporated into Applesoft programs.
Price: $25 includes system diskette (DOS 3.2) with sample program and documentation manual.
Author: **Robert F. Zant**
Available: **Decision Systems**
P.O. Box 13006
Denton, Texas 76203

Name: **'DUSC' — Disk Utility Sorted Catalog**
System: Apple II, or Apple II Plus
Memory: 32K
Language: 6502 machine code
Hardware: Apple II with Disk II
Description: A machine language utility for sorting the disk catalog. Works with any version of DOS (3.1/3.2/3.2.1/3.3). Physically reorders the catalog entries on the disk into ASCII collating sequence. Fast, can sort full catalog in less than 10 seconds. Preserves delete indicator integrity and forces deleted files to sort last. Complete error handler. Interfaces to user in a manner similar to UPDATE 3.2.
Copies: Just Released
Price: $15.95 includes disk and documentation.
Author: **Richard E. Rettke**
Available: **RER Software**
1757 Acorn Ct.
Menasha, WI 54952

Name: **Mind Bogglers I**
System: Atari 400 or 800
Memory: Cassette version 16K, Disk 24K
Language: BASIC
Hardware: Atari 400 or 800 with 410 cassette recorder or 810 disk drive
Description: Consists of 3 strategy games: Capture (based on Othello™), Mystery Box and Simon Says. Capture is a strategy game in which you and your computer fight for control of the board. In Mystery Box the player shoots rays into the box to find the hidden atoms. Simon Says is a memory teaser in which you must repeat the computer's pattern.
Price: $15.95 cassette, $19.95 disk
Author: **Gary R. See**
Available: **Versa Computing, Inc.**
887 Conestoga Circle
Newbury Park, CA 91320

MICRO
6502 Bibliography: Part XXIX

834. Compute 1, Issue 5 (July/August, 1980)

Moshell, J.M., "Assembly Language Programming with UCSD PASCAL," pg. 52-57.
Tutorial including a program to make low-resolution 16-color graphics available to Pascal.

Victor, John, "Adding a Voice Track to Atari Programs," pg. 59-61.
Discussion and Listing for the Atari.

Fortner, Charles G., "The Basics of using POKE in Atari Graphics," pg. 62.
Discussion with a simple program and an expanded program by Robert Lock.

Harris, Neil, "Color Wheel for the Atari," pg. 64.
A simple graphics program for the Atari system.

Lindsay, Len, "Choose Your Joystick," pg. 64.
It takes only a few lines in BASIC to allow you to use whatever joystick plug you wish.

Isaacs, Larry, "Input/Output on the Atari," pg. 65-68.
A tutorial on how to use BASIC commands to communicate with your peripheral devices.

Butler, Brett, "Un-New," pg. 81.
While NEW erases a program from memory, UN-NEW will help you recover that program, by rebuilding the chain and restoring the variable pointers, for the PET.

Poirier, Rene W., "Disk ID Changer," pg. 83.
A PET listing for changing either the name or the ID number on a diskette.

Butterfield, Jim, "Shift Work," pg. 85.
Tricks with the PET's shift key.

Wollenberg, Robert H., "Machine Language Code for Appending Disk Files," pg. 86.
A useful tool for the PET DOS to help in appending programs.

Butterfield, Jim, "Mixing BASIC and Machine Langauge," pg. 89-90.
Put BASIC and Machine Language together on your PET, with several examples: Universal ROM Test; RAM Test; Tape Test; Leader Write.

Busdiecker, Roy, "After the Monitor's Moved," pg. 92-93.
An instructional article for the PET.

Butterfield, Jim, "Fitting Machine Language into the PET," pg. 94-95.
A tutorial article on the PET memory management.

Herman, Harvey B., "Joystick Revised," pg. 99-100.
A machine language for the PET that allows one to use joysticks, without any changes in programs.

835. The Target (July/August, 1980)

Butterfield, Jim, "Interfacing AIM BASIC," pg. 2-3.
An instructional article on marrying AIM BASIC to machine language.

Hall, Dale, "BASIC Bandaid," pg. 4.
Improvements in the AIM 65 tape loading routine.

Rathbun, Michael, "AIM Display," pg. 7.
A quick-and-dirty slow display for AIM systems not using a teletype.

Silber, Steve, "Directory," pg. 8-9.
A program for the AIM designed to inventory the contents of AIM 65 format tapes.

836. Call—A.P.P.L.E. 3 No. 6 (July/August, 1980)

Howard, Clifton M., "Directory Title Formatting," pg. 7-23.
A major directory utility for the Apple II.

Reynolds, Lee, "Types of Memory Moves," pg. 25-27.
A tutorial on moving machine language programs around in the Apple II memory.

Anson, Christopher P., "Trix to PEEK and POKE in Pascal," pg. 28-29.
A utility to allow Pascal programmers to PEEK and POKE at a location without the need for an assembly language program linked to the primary program in the Apple.

Huelsdonk, Bob, "Making BASIC Behave, Part IV," pg. 33-35.
Continuing the development of a program to enter and store data on the Apple disk.

Anon., "Converting 'Reconstruct VTOC' for Apple II Plus," pg. 35.
This useful utility for the Apple II has now been converted for use on the Apple II Plus.

Golding, Val, "A Subroutine Becomes a Program," pg. 43.
An alphabetizing program for the Apple.

DeGroat, Ron, "Inverse and Flashing Modes for Apple Pascal", pg. 46.
An assembly language listing for Pascal Users.

Golding, Val J., "Applesoft Input Nearly Anything Subroutine," pg. 47-48.
This program allows inputting a string in Applesoft which contain items like a semicolon or quotation marks.

Huntress, Wes, "Hi-Res Screen Switch," pg. 48-49.
An Apple machine language program to move a scene from one Hi-Res page to another.

Golding, Val J., "How to Have Your Cake and Eat It Too," pg. 50-52.
A group of Integer BASIC programs for the Apple.

Carner, Douglas, "Beyond Integer BASIC," pg. 52-53.
Applesoft functions duplicated in Integer BASIC for the Apple.

Boody, Charles, "Comparing Applesoft Programs for Differences," pg. 54-56.
Two listings that will assist you in ferreting out the changes that have been made in a modified program.

Flanagan, Dale, "Adding Cursor Control to Aptype," pg. 58.
Four added lines will make your Aptype program more versatile.

837. MICRO No. 27 (August, 1980)

Brady, Virginia Lee, "Data Statements Revisited," pg. 7-11.
The fundamentals of the technique and examples of a program to update program statements (for Applesoft).

MacCluer, C.R., "Satellite Tracking with the AIM-65," pg. 13-14.
An astronomy program for the AIM 65, but easily adapted to other 6502 systems.

Chipchase, Frank D., "Better Utilization of Apple Computer Renumber and Merge Program," pg. 17-18.
Here is a technique to enhance the utility of the useful Renumber and Merge program.

Cadmus, Ray, "Variable Lister," pg. 19.
This program will extract the variable names from your Basic Apple program and sort and list them.

Golla, Lawrence R., "Nth Precision Add and Subtract with Adjusted Processor Status," pg. 27-29.
A general utility program for the 6502 family.

Partyka, David A., "Solar System Simulation With or Without an Apple II," pg. 33-39.
Apple Graphics and the laws of the universe combine to make a super demonstration.

Taylor, William L., "Interface of OSI-C1P with Heath Printer," pg. 47-51.
Hardware and Software to implement the Heath H-14 printer with OSI machines.

Motola, R.M., "Applesoft Floating Point Routines," pg. 53-55.
A discussion of some important Applesoft routines and their use.

Beamsley, Jeff, "Up From the Basement," pg. 59.
Discussion of OSI microcomputer information resources.

Finn, Kenneth, "Son of Screen Print," pg. 61-63.
A mini-word processor for the PET.

Bauers, Barton M., Jr., "Business Dollars and Sense in Applesoft," pg. 65-67.
Rounding and Formatting business data.

Soltero, Richard, "BCD Input to a 6502 Microprocessor," pg. 68-70.
A basic program for inputting data to 6502 machines like the SYM-1 or AIM-65.

Rowe, Mike (Staff), "The MICRO Software Catalog; XXIII," pg. 71-73.
Seventeen new software items for 6502 micros are reviewed.

Dial, William R., "6502 Bibliography: Part XXIII," pg. 75-77.
About 120 new references for the 6502 users.

838. Dr. Dobb's Journal 5, Issue 7, No 47 (August, 1980)

Kruse, Richard M., "ZX65: Simulating a Micro," pg. 4-10.
An assembly language program which simulates a 6502 using a Z-80 board.

839. The Apple Barrel 3, No. 6 (August, 1980)

Black, David C., "Pascal Tutorial-Lesson 2," pg. 5-8.
Continuation of a tutorial started several months ago. With several demo listings, for the Apple.

Meador, Lee "DOS Disassembly," pg. 11-16.
A continuation of this series on the DOS 3.2 for the Apple.

840. OSI Users Independent Newsletter, No. 5 (Aug. 1980)

Hooper, Philip K., "Command Decoding in 65D," pg. 1-2.
Discussion plus decoding table for OSI users.

Curley, Charles, "Clear and Color Screen Routine," pg. 3.
A quick screen clear routine for the OSI user.

Geoffroy, John, "File Directory Re-Creation Utility," pg. 5-6.
A program to restore zapped directories on OSI systems.

841. The Harvest 1, No. F (August, 1980)

Pump, Mark, "Mark Pump's DOS Patches," pg. 1-5.
Changes in DOS 3.2, 3.2.1 to allow faster INIT and other functions and lots of other useful mods.

Stadfeld, Paul, "Space Exploration," pg. 8.
All about the little known SPC function on the Apple.

Lundeen, Rich, "Character Poking," pg. 12.
Experiment with poking characters to the Apple screen.

842. Creative Computing 6, No. 8 (August, 1980)

McBurney, N.B. II, "Apple Pie," pg. 110-113.
Understanding pie charts and the Apple high resolution graphics in plotting charts.

Mechner, Jordan, "Translating Into Apple Integer Basic," pg. 124-125.
All about the differences between an Integer basic and floating point type basics.

Wolff, Bruno B. Jr., "Apple II: Reading Data From Tape," pg. 126-127.
A tutorial for the Apple II.

Carpenter, Chuck, "Apple Cart," pg. 148-152.
Discussion of Disk Drives, Apple III, Super-Text word processor, and a Pascal program and an integer string program.

Blank, George, "Outpost: Atari," pg. 154-156.
Differences between Atari Basic and various Microsoft Basics, Sound programs, Text Handling, Jumps and Subroutines, I/O routines, etc. on the Atari.

843. Byte 5, No. 8 (August, 1980)

Williams, Gregg, "The Ohio Scientific CA-15 Universal Telephone Interface," pg. 40-44.
An interface under software control that puts your OSI micro on the telephone line.

844. Softside 2, No. 11 (August, 1980)

Blank, George, "Converting Graphics from One Computer to Another," pg. 22-23, 88.
A comprehensive look at the graphics of the Apple, Atari and TRS-80 computers.

Smith, Bill, "ROM the ROBOT," pg. 42-44.
Part 3 of a continuing graphics program for the Apple.

Anon, "You Can Have Sound on Your Computer," pg. 66-69.
Listings for Apple, Atari and TRS-80.

845. The Cider Press (March, 1980)

Anon, "Disk of the Month—March '80," pg. 1.
Fourteen programs on an Apple Disk.

Bernheim, Phil, "Read Those Mysterious 'T' Files," pg. 5.
An easy way to read files on the Apple.

Pfeifer, Frank, "Notes on Disk Aide," pg. 6.
Notes on this Useful Utility, for the Apple.

Chipchase, Frank, "Better Use of Apple II Renumber and Merge Program," pg. 7.
Suggestions for improving the use of this Apple Utility.

Anon, "Disk of the Month," pg. 3.
Twenty one listings on an Apple Diskette.

Anon, "Utilities," pg. 6.
Tricks with the Parallel Printer Card for the Apple, Apple tricks with the DOS, a Gasoline price conversion program, etc.

Wozniak, Steve, "Binary-To-Decimal Shortcut," pg. 7.
A Shortcut by one who should know his way around the Apple.

846. Robert Purser's Magazine, Edition 9 (Summer, 1980)

Purser, Robert, "Software Directory: Apple II, TRS-80 level II, Atari."
Reviews of programs and a comprehensive list of available software.

847. KB Microcomputing, No. 44 (August, 1980)

RE', Ugo V., "Improving the OSI Challenger C2," pg. 40-46.
First in a two-part series on the inner workings of OSI "500" boards.

Pytlik, William F., "PET I/O Port Expander," pg. 96-98.
A description of Joystick interfacing to the PET.

Moore, Robin B., "Graphics Character Generator," pg. 106-117.
Mix text and graphics anywhere on the Apple II screen with this CHAR-GRAF program.

Moore, William R., "Let PET Design Your Next Power Supply," pg. 130-133.
Make your PET an Engineer/Draftsman with this program.

Yob, Gregory, "Get Your PET on the IEEE 488 Bus," pg. 134-140.
This second part of the series examines the file characteristics of the IEEE 488 bus.

Boynton, G.R., "A 'Personable' Calendar" pg. 168-171.
Let your PET clear up confusion in your lists.

848. The Seed 2, No. 8 (August, 1980)

Davis, William, "Empty Files," pg. 6.
How to create an empty file designation on the Apple Catalog.

Greene, Amos, "Renumbering Disk Volumes," pg. 9.
A simple method to change the Volume No. on the Apple Disk.

White, Harry C., "Russian Roulette with RND," pg. 10.
All about Random numbers on the Apple.

Smith, W.B., "A Shape Maker for Apple II Lo-Resolution Color Graphics," pg. 14.
Create shapes for your games.

Wise, Bruce, "REM Stripper," pg. 17-18.
A routine for the Apple which removes all REM statements from a program for faster operation.

Hance, E., "Apple Tape Beeps Translated," pg. 19.
Explanation of those beeps on the Apple Tape format.

Crossman, Craig, "Fun With Assembler," pg. 20-22.
An assembly Language tutorial for the Apple II.

Rivers, Jerry, "Sorts," pg. 23-26.
A sorting tutorial for the Apple.

849. Interface Age 5, Issue 8 (August, 1980)

MacDougall, John, "Put a Daisy on Your Apple," pg. 76-78.
Describes an interface for adding a Diablo Hytype-1 or a Qume Sprint Micro 3 to the Apple System.

Zant, R.F., "Formatting Integer Basic Programs," pg. 96-99.
Use of Asterisks in formatting program listings.

850. River City Apple Corps Newsletter (August, 1980)

Bartley, David H., "Rational Approximations for Floating Point Numbers," pg. 5-7.
MAKRAT is an Applesoft program to demonstrate an ancient mathematical algorithm.

851. Southeastern Software Newsletter, Issue 20 (August, 1980)

Staff, "Setting Character Sizes for Integral Data Printers," pg. 1-3.
A machine language routine for the Apple.

Staff, "Binary Converter," pg. 3-6.
A short routine for the Apple.

Hartley, Tim, "Tim Hartley and The Extra Catalog Track," pg. 5-7.
Here is a short program to allow 184 file names in the Apple catalog instead of the usual 84 max.

Hartley, Tim, "DOS Removal," pg. 5-7.
DOS OFF is a short program for removing the DOS from any Apple Diskette, thus opening up about 6.5K extra space.

Reich, Leo "Recovery of Apple Basic Programs after Executing 'New' " pg. 8.
Recover from an otherwise costly error with this one.

852. PEEK(65) 1, No. 8 (August, 1980)

Sanders, James H. "Long String Input for OSU Systems," pg. 4-5.
A program for OSI Micros.

Showalter, Bruce, "Cheapie Superboard II Expansion," pg. 9-14.
A hardware article for The Superboard II user.

Shingara, Terry, "Baud rate Mod," pg. 18.
A Hardware Mod for OSI 600 boards.

Badger, Robert, "Modified Garbage Collector," pg. 19-20.
A compact version of a method of correcting string handling problems on the OSI system.

Anon, "Card Shuffling Routine, pg. 22.

853. G.R.A.P.E. (August, 1980)

DeGroat, Ron, "Inverse & Flashing Modes for Apple Pascal Text Display," pg. 3.
Assembly Language Listings for Pascal.

Anon, "Polar Function Graphs," pg. 4.
Two demonstration listings for Apple Hi Res.

854. Abacus II 2, Issue 8 (August, 1980)

Freeman, Larry, "Writing to Disk from a Machine Language Program," pg. 3-4.
How to give commands such as OPEN, READ, Write, etc. from a machine language program.

Anon, "IAC Application Note: DOS 3.2 Demo Program," pg. 8-12.
Six programs in Applesoft.

Freeman, Larry, "The Great Applewriter Interface," pg. 13-22.
A major set of Utilities for interfacing Apple Writer and Basic Integeror Applesoft listing.

855. From the Core (August, 1980)

Budge, Joe, "Base Two User Note," pg. 4.
Using the SSM board to interface the Apple with a Base 2 printer at 19,200 Baud.

856. Personal Computing 4, No. 8 (August,1980)

Derner, Robert R., "A Teacher for Your Apple," pg. 48-49.
A flash card type program for the Apple.

Vann, Eric Geoffrey, "A Gradebook for Teacher," pg. 53-63.
A science grading program and administrative aid, for the Apple.

Anon, "Apple III Unveiled at NCC," pg. 65-67.
Description of the latest Apple System.

857. Fort Worth Apple User Group Newsletter (FWAUG) 2, No. 1 (August, 1980)

Lyons, George, "Integer Basic Text Editor," pg. 2-7.
A simple text editor operating on text instead of code.

Neiburger, Skip, "Computer Disks," pg. 8-10.
Comparison of various brands of diskettes in copy programs, rating reliability shows marked variations.

Meador, Lee, "DOS Disassembly 7," pg. 10-15.
This installment of this important series includes the command decoder section of DOS (Apple).

858. Applesass (August/September, 1980)

Staff, "Primer of the 3.3 DOS - Meet Muffin," pg. 7.
A discussion of advantages and problems associated with the new DOS for Apple.

Anon., "One Liners," pg. 11.
Herringbone and honeycombs in Hi-Res on the Apple.

859. Rubber Apple Users Group Newsletter 3, No. 6 (August/September, 1980)

Staff, "Patch for Dakin 5 Programming Aids II," pg. 1.
A patch for Dakin 5 allowing the PATCHER to work on DOS 3.3 diskettes.

Throop, Gil, "Rapidly Transfer Data Between Arrays and Disk," pg. 2-5.
A program for the Apple.

Gabelman, Ken, "Disk Structures II," pg. 6-9.
Continuing discussion and listings for Random Data Files.

860. Stems From Apple 3, No. 8/9 (August/Sept., 1980)

Rivers, Jerry, "Technical Tidbits: The Great DOS Append Fix," pg. 12.
A fix for the Apple DOS Append and a fix for the fix.

Anon., "Dollars and Cents," pg. 16.
A formatting program for the Apple.

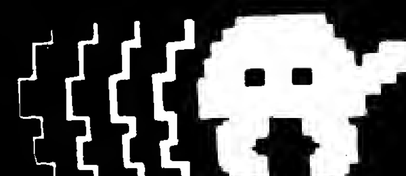
Evans, Frank, "Using USR," pg. 17-19.
Tutorial on the USR function for the Apple.

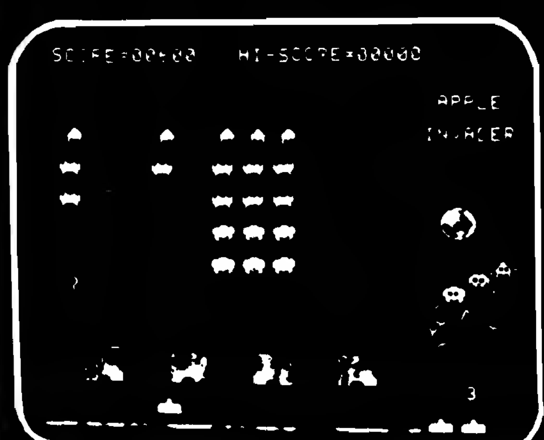
SPACE WAR

You're in command in **SPACE WAR!** Destroy your opponent's ship by forcing him to collide with the sun or to explode upon re-entry from hyperspace ... or challenge him face to face with missile fire. You're in command of the speed and direction of your ship. You control the timing of your missiles. You select the game mode from five options, including Reverse Gravity, and the battle begins. Accelerate to place your shots--and escape into hyperspace before your opponent comes within range. But be wary, he (or she!) may circle out of sight and reappear on the opposite side of the galaxy! (This is the classic MIT game redesigned especially for the Apple.)

and SUPER INVASION

- **Super Invasion** is the original invasion game, with the original moon creatures and faster action than any other invasion game.
- Features superb high resolution graphics, nail-biting tension and hilarious antics by the moon creatures!
- Self-running "attract mode" of operation for easy learning and demonstrating of the game.
- As good in every way as the famous Invaders arcade game.
- High speed action • Sound effects!
- Runs on the Apple II and the Apple II Plus





Fifty-five aliens advance and shower you with lethal writhing electric worms. As you pick off the aliens, one-by-one, they quicken their descent. They whiz across the screen wearing away your parapets, your only defense, coming closer and closer to your level. **Super Invasion** is the **original** invasion game with the original moon creatures and faster action than any other invasion game on the market.

Super Invasion is available for only $19.95 on cassette (CS-4006) for a 32K Apple II. **Space War** is $14.95 on cassette (CS-4009) for a 16K Apple II. **Space War** and **Super Invasion** are on one disk (CS-4508) for a 48K Apple II for only $29.95.

Send payment plus $1.00 shipping and handling to Creative Computing Software, P.O. Box 789-M, Morristown, NJ 07960. NJ residents add $1.00 sales tax. Bankcard orders may be called in toll free to 800/631-8112. In NJ call 201/540-0445.

sensational software

creative computing software

ADVERTISERS' INDEX

February 1981

The home computer you thought was years away is here.

C8P DF

Ohio Scientific's top of the line personal computer, the C8P DF. This system incorporates the most advanced technology now available in standard configurations and add-on options. The C8P DF has full capabilities as a personal computer, a small business computer, a home monitoring security system and an advanced process controller.

Personal Computer Features

The C8P DF features ultra-fast program execution. The standard model is twice as fast as other personal computers such as the Apple II and PET. The computer system is available with a GT option which nearly doubles the speed again, making it comparable to high end mini-computer systems. High speed execution makes elaborate video animation possible as well as other I/O functions which until now, have not been possible. The C8P DF features Ohio Scientific's 32 x 64 character display with graphics and gaming elements for an effective resolution of 256 x 512 points and up to 16 colors. Other features for personal use include a programmable tone generator from 200 to 20KHz and an 8 bit companding digital to analog converter for music and voice output, 2-8 axis joystick interfaces, and 2-10 key pad interfaces. Hundreds of personal applications, games and educational software packages are currently available for use with the C8P DF.

Business Applications

The C8P DF utilizes full size 8″ floppy disks and is compatible with Ohio Scientific's advanced small business operating system, OS-65U and two types of information management systems, OS-MDMS and OS-DMS.

The computer system comes standard with a high-speed printer interface and a modem interface. It features a full 53-key ASCII keyboard as well as 2048 character display with upper and lower case for business and word processing applications.

Home Control

The C8P DF has the most advanced home monitoring and control capabilities ever offered in a computer system. It incorporates a real time clock and a unique FOREGROUND/BACKGROUND operating system which allows the computer to function with normal BASIC programs at the same time it is monitoring external devices. The C8P DF comes standard with an AC remote control interface which allows it to control a wide range of AC appliances and lights remotely without wiring and an interface for home security systems which monitors fire, intrusion, car theft, water levels and freezer temperature, all without messy wiring. In addition, the C8P DF can accept Ohio Scientific's Votrax voice I/O board and/or Ohio Scientific's new universal telephone interface (UTI). The telephone interface connects the computer to any touch-tone or rotary dial telephone line. The computer system is able to answer calls, initiate calls and communicate via touch-tone signals, voice output or 300 baud modem signals. It can accept and decode touch-tone signals, 300 baud modem signals and record incoming voice messages.

These features collectively give the C8P DF capabilities to monitor and control home functions with almost human-like capabilities.

Process Controller

The C8P DF incorporates a real time clock, FOREGROUND/BACKGROUND operation and 16 parallel I/O lines. Additionally a universal accessory BUS connector is accessible at the back of the computer to plug in additional 48 lines of parallel I/O and/or a complete analog signal I/O board with A/D and D/A and multiplexers.

Clearly, the C8P DF beats all existing small computers in conventional specifications plus it has capabilities far beyond any other computer system on the market today.

C8P DF is an 8-slot mainframe class computer with 32K static RAM, dual 8″ floppies, and several open slots for expansion.

Prices start at under $3,000.

Computers come with keyboards and floppies where specified. Other equipment shown is optional.

For literature and the name of your local dealer, CALL 1-800-321-6850 TOLL FREE.

OHIO SCIENTIFIC
a M/A-COM Company
1333 SOUTH CHILLICOTHE ROAD
AURORA, OH 44202 • (216) 831-5600